



H446/03

Callum Tarttelin, 1164, 51617

Table of Contents

1. Research	1
1.1. Emails	1
1.1.1. The Email	1
1.1.2. Background.....	1
1.1.3. Handicapping.....	1
1.1.4. Scoring.....	2
1.1.5. League Standings	3
1.2. Next stuff.....	4
2. Planning	4
2.1. What objects, and what do they have	4
2.2. Database modelling	5
2.3. Api planning	6
2.4. Technologies	6
2.5. Design	7
2.6. Research	7
2.7. Testing	7
2.8. Ordering	7
3. Development	7
3.1. Initial Setup	8
3.2. Modelling The Objects	11
3.2.1. League.java	11
3.2.2. Team.java	11
3.2.3. Player.java	12
3.2.4. Game.java	13
3.2.5. Errors at this stage	14
3.3. Repositories	14
3.3.1. Errors at this stage	14
3.4. Controllers	15
3.4.1. index.ftl	15
3.4.2. HomeController.java	16
3.4.3. Errors at this stage	16
3.5. League Controller	17
3.5.1. LeagueControllerTest.java	17
3.5.2. LeagueController.java	20
3.5.3. LeagueRepository.java	23
3.5.4. LeagueView.java	23
3.5.5. LeagueViewSummary	24
3.5.6. Errors at this stage	25

3.6. Front End Setup	26
3.6.1. Npm Setup	26
3.6.2. webpack.config.js	27
3.6.3. Babel	28
3.6.4. Errors At This Stage	28
3.7. Front End for leagues	28
3.7.1. Dependencies	29
3.7.2. app.js	29
3.7.3. LeagueSummary.js	30
3.7.4. LeagueList.js	31
3.7.5. League.js	33
3.7.6. AddLeague.js	34
3.7.7. Delete.js	35
3.7.8. Errors at this stage	38
3.8. Team Controller	40
3.8.1. TeamControllerTest.java	40
3.8.2. TeamRest.java	42
3.8.3. TeamController.java	42
3.8.4. TeamView.java	44
3.8.5. TeamViewSummary.java	46
3.8.6. Errors at this stage	46
3.9. Team front end	47
3.9.1. League.js	47
3.9.2. App.js	47
3.9.3. AddTeam.js	47
3.9.4. TeamSummary.js	49
3.9.5. Team.js	49
3.9.6. Errors at this stage	51
3.10. Nightwatch Testing	51
3.10.1. nightwatch.json	51
3.10.2. TestLeague.js	52
3.10.3. TestTeams.js	54
3.10.4. Tests at this point	56
3.11. 1. Teams Work	56
3.11.1. JUnit Tests	56
3.11.2. com.saskcow.bowling.repository.GameRepositoryTest	56
3.11.3. com.saskcow.bowling.repository.LeagueRepositoryTest	56
3.11.4. com.saskcow.bowling.BowlingApplicationTests	56
3.11.5. com.saskcow.bowling.controller.TeamControllerTest	57
3.11.6. com.saskcow.bowling.controller.LeagueControllerTest	57
3.11.7. Nightwatch Tests	57

3.11.8. TestTeams	57
3.11.9. TestTeams	57
3.11.10. TestLeague	60
3.11.11. TestLeague	60
3.11.12. Errors at this stage	63
3.12. Player Controller	63
3.12.1. PlayerControllerTest.java	64
3.12.2. PlayerController.java	65
3.12.3. PlayerRest.java	67
3.12.4. PlayerView.java	67
3.12.5. PlayerViewSummary.java	68
3.12.6. Errors at this stage	68
3.13. 2. Players back end only	69
3.13.1. JUnit Tests	69
3.13.2. com.saskcow.bowling.repository.GameRepositoryTest	69
3.13.3. com.saskcow.bowling.repository.LeagueRepositoryTest	69
3.13.4. com.saskcow.bowling.BowlingApplicationTests	69
3.13.5. com.saskcow.bowling.controller.TeamControllerTest	69
3.13.6. com.saskcow.bowling.controller.LeagueControllerTest	70
3.13.7. com.saskcow.bowling.controller.PlayerControllerTest	70
3.14. Player Front End	70
3.14.1. App.js	70
3.14.2. Team.js	71
3.14.3. AddPlayer.js	71
3.14.4. PlayerSummary.js	72
3.14.5. Player.js	73
3.14.6. TestPlayers.js	75
3.15. 3.Players Front and Back	77
3.15.1. Nightwatch Tests	77
3.15.2. TestPlayers	77
3.15.3. TestPlayers	77
3.15.4. TestTeams	80
3.15.5. TestTeams	80
3.15.6. TestLeague	83
3.15.7. TestLeague	83
3.16. Games in Back end	85
3.16.1. GameControllerTest.java	85
3.16.2. GameController.java	88
3.16.3. GameRest.java	90
3.16.4. GameView.java	90
3.16.5. Errors at this stage	91

3.17. 4. Games in back end	91
3.17.1. JUnit Tests	91
3.17.2. com.saskcow.bowling.repository.GameRepositoryTest	91
3.17.3. com.saskcow.bowling.repository.LeagueRepositoryTest	92
3.17.4. com.saskcow.bowling.BowlingApplicationTests	92
3.17.5. com.saskcow.bowling.controller.TeamControllerTest	92
3.17.6. com.saskcow.bowling.controller.LeagueControllerTest	92
3.17.7. com.saskcow.bowling.controller.PlayerControllerTest	93
3.17.8. com.saskcow.bowling.controller.GameControllerTest	93
3.18. Games in the front end	93
3.18.1. App.js	93
3.18.2. League.js	94
3.18.3. AddGame.js	94
3.18.4. GameSummary.js	98
3.18.5. Errors at this stage	98
4. Stage 2	99
4.1. Planning	99
5. Test Formatter	99
5.1. Research	99
5.2. Planning	99
5.3. Development	99
5.4. Testing	99
6. Testing	99
7. Api Docs	99
7.1. League API	100
7.1.1. Create A League	100
7.1.2. Get a list of Leagues	100
7.1.3. Get one League	100
7.1.4. Get a list of leagues by name	101
7.1.5. Delete A league	101
7.2. Team API	101
7.2.1. Create A Team	101
7.2.2. Get A Team	102
7.2.3. Delete A Team	102
7.3. Player API	102
7.3.1. Create A Player	102
7.3.2. Get A Player	103
7.3.3. Delete A Player	103
7.4. Game API	103
7.4.1. Create A Game	103
7.4.2. Get A Game	104

7.4.3. Delete A Game	104
7.4.4. Complete A Game	104
7.5. PlayerGame	104
7.5.1. Create 3 PlayerGames	105
7.5.2. Delete A PlayerGame	105
7.6. Score API	105
7.6.1. Create A Score	105
7.6.2. Delete A Score	106

1. Research

The client currently has a bowling league with an older website. The purpose of this project will be to replace this website with a website easier to use.

1.1. Emails

I received an email from the client about what the project will need to do.

1.1.1. The Email

Welcome to the seemingly easy world of tenpin bowling league management

All of this is currently managed through a number of Excel worksheets linked together, with a load of macros to produce the static web pages and the scoresheets for each week with the updated averages, handicaps and league standings. The web pages are then uploaded to the website each week. It would be great to get to a point where the data entry can be done directly into a [database] on the web server so it can be done from anywhere (and in theory, by anyone). This would help when I am on holiday. Then we can look at webpages that don't need to be uploaded, they would simply get data out of the database. I think it would make sense to meet up so I can show you the mess I work with (which causes headaches at the start of each season when I tweak it to fit – and inevitably break something). That way you'll also be able to ask "why on earth do you do that", or "you never mentioned that". Maybe I can show you how I setup a new season, which is when I really find out the bits I need!

1.1.2. Background

The basic structure is that we have a number of teams competing over a number of weeks which make up a season. Each team can have up to 9 bowlers registered and actively bowling. A bowler can switch from one team to another mid-season, however they can only move once during the season. When this happens, their average and handicap move with them. This happens rarely but does happen.

1.1.3. Handicapping

The league runs a handicapping system providing additional points to a bowler based on their current average score. The calculation that we use in Excel is =INT(200-INT(bowler_average*0.8) There is a maximum handicap of 80 and a minimum of 0. A bowler with an average of 180 has a handicap of 16 ((200-180)*0.8), an average of 180.9 is the same. A bowler with an average of 90 has a handicap of 80 ((200-90)*0.8) = 88 (more than the maximum) A new bowler joining the league will not have an average, hence no handicap; they will receive a handicap based on their first night's scores. So, if Lucy, in the example below, had bowled those scores on the first night, her handicap would have been 64 based on her average of 120. This then is applied to her scratch scores – she would have won her individual match 6-2 instead of losing 3-5! To keep the handicap current, we use the last 24 scratch game scores to calculate the average for each bowler. Just to be confusing, we do also note the average for all league matches through the season (this is used to determine the "high season average" award – but enough of awards! The "blind" score is simply calculated by adding the bowler's average to their handicap and rounding down... average of 120 = handicap of

64 = blind score of 164 which they would need to beat to win points if the opponent didn't turn up.

1.1.4. Scoring

Each week 3 bowlers per team take part and play 3 games each. In reality, it is more complicated or flexible than this. It could be that each game is played by a different bowler, in the event people get injured; so we need the flexibility for each game to be attributed to a different bowler with a different handicap. Example scoresheet (from the website)

Mid Lane Crisis									11
HC P	Bowler	Game 1		Game 2		Game 3		Total	Pts
58	Lucy Scott	131	189	124	182	105	163	360	534
		2 pts	0 pts	0 pts	0 pts	1 pts			3
36	Keith Biggs	177	213	179	215	168	204	524	632
		0 pts	2 pts	0 pts	0 pts	0 pts			2
43	Simon Taylor	159	202	138	181	98	141	395	524
		2 pts	0 pts	0 pts	0 pts	0 pts			2
		467	604	441	578	371	508	127	1690
		2 pts	2 pts	0 pts	0 pts	9			4

Spare Us									21
HC P	Bowler	Game 1		Game 2		Game 3		Total	Pts
55	David Henn	108	163	131	186	130	185	369	534
		0 pts	2 pts	2 pts	2 pts	1 pts			5
52	Mihir Sampat	180	232	117	169	189	241	486	642
		2 pts	0 pts	2 pts	2 pts	2 pts			6
50	Mark Brosnan	134	184	168	218	142	192	444	594
		0 pts	2 pts	2 pts	2 pts	2 pts			6
		422	579	416	573	461	618	129	1770
		0 pts	0 pts	2 pts	2 pts	9			4

In the match above Lucy Scott is competing directly against David Henn (first bowler for each team). Their individual match is made up of 4 elements; game 1, game 2, game 3 and the series. I currently record the scratch score (131 for Lucy and 108 for David), their handicap and the bowler name. Once the handicap is added to the scratch score we get the handicapped score, this is what is compared to determine who won the points. In this case Lucy's handicapped score is 189 which beats David's handicapped score of 163, so Lucy is awarded 2 points for game 1. The points are awarded for games 2 and 3 and finally for the series totals. In this instance because the scores are tied (534) they both are awarded 1 point. In theory, we could calculate the handicap "on the fly", however recording the handicap as a static value it allows for the anomalies we see at various times during a season (postponement of matches). Score is each 2 points for a win, 1 point for a

draw (equal score). 32 points are available each week: Each bowler 2 points per game (3 games) plus 2 points for the total (series) score with handicap. Scored against their equivalent bowler on the opposite team (based on who bowls 1st/2nd/3rd). So each bowler can score up to 8 points. The team total counts as another pseudo-bowler and is scored the same way – total per game and grand-total. The handicap score (meaning the bowlers actual or “scratch” score plus their handicap) is always used to work out who won/lost/tied.

1.1.5. League Standings

Position	Team	Games	Pins For	Pins Against	HHG Season	HHG All	HHS Season	HHS All	Team Pts	Total Pts
1	Tigers	78	44736	43736	698	698	1903	1903	141	508
2	Strike Force	78	43972	43755	656	656	1802	1802	119	445
3	Hook Line & Sinker	75	42535	42024	662	662	1897	1897	119	432
4	Just Good Friends	78	44855	44592	718	718	1933	1933	108	428
5	Spare Us	78	44057	43603	673	673	1821	1821	111	425
6	Jets	78	44221	43998	650	667	1852	1877	108	422
7	Mid Lane Crisis	78	44005	44463	667	667	1876	1876	111	397
8	Raiders	78	43812	44153	645	645	1797	1797	98	371
9	Razors	78	43557	44657	667	667	1827	1827	84	366
10	Easy Does It	75	41586	42355	684	684	1985	1985	73	304

The league table page shows

- **HHG*** = High Handicap Game (the total team score not per bowler)
- **HHS*** = High Handicap Series (series = sum of the 3 games played any week, again for the team total)
- **Pins for** = total of the scratch scores scored by the team to date
- **Pins against** = total of their oppositions scores each week to date
- **Team Pts** = total points won by the team pseudo bowler each week
- **Total Pts** = all points won by the team
- *There is a “Season” and “All” value for each of these – based on the fact we run a cup competition each season.

That covers the basics of the scoring and how points are allocated. From all of this, I provide a large number of stats (because I am go through phases of being interested in it).

The Cup

The cup competition that runs for a few weeks during the year is also managed this way – but more as an afterthought. The mechanism for determining points can be different to the normal league depending upon how we manage it; normally it's different when we have an odd number of teams. For each team we see a summary of the score each week and some individual achievements. There is a great deal of detail (which I have put together because it was available)... we can make this “phase 2” Ian’s comments I haven’t addressed elsewhere

Some weeks teams will agree to postpone their game. They catch up the missed games later in the season

- This is where the static handicap is useful, with the option to override it. Bowler's stats are rolled over between seasons (so they continue with the average from last season at the start of a new season)
- Their last 24 games are used to roll their average to the new season. If they haven't bowled 24 games, then up to 24 games are carried over; if it's less than 6 games then they will start as if new. All achievements are carried over (awards for a 200 game, or 6 consecutive strikes)
There could be a different number of teams some seasons
- If it's an odd number the league make the decision to either bowl against the "blind", or to have a week of no bowling. A team could drop out mid-season and scores must be removed from each weeks results (is that right Phil?)
- It depends, I will check the constitution; if they have bowled against each team just once but a couple of teams twice, we might remove the extra couple of results. We normally encourage them to bowl to a natural break point, but it's not always possible. At the start of the season the system needs to generate a list of matches – which team is playing which other team and on which lanes. The teams need to be put on each lane an equal number of times as far as possible and play each other team twice (maybe 3 times if the number of teams is low enough and there are enough weeks available). If the league has 10 teams assume lanes 1 to 10 are used each week
- We are typically a 10 team league and to be awkward we use lanes 3 to 12 but if we have 10 lane identifiers, we can modify them to be the lanes we actually use.
Some weeks are used for non-league matches (special competitions as teams or individuals tournaments). Some weeks each summer there is no bowling (too maybe people on holiday so we stop). Also bank holidays there is no playing.
- We need a mechanism for recording these scores (we currently use a different area of the spreadsheet to record these, so they are not included within the range used for calculating averages and handicaps but within the range used for awards).

1.2. Next stuff

On top of this existing criteria the current project should have a login system to allow users to add scores for their games. This will need to be confirmed by the other team.

2. Planning

2.1. What objects, and what do they have

```

League{
    teams: [Team, Team ...],
    rota: [Game, Game ...]
    ranking: computed
}

Team{
    name: String
    Image: Image
    players: [Player, Player ...]
    score: Score Object
}

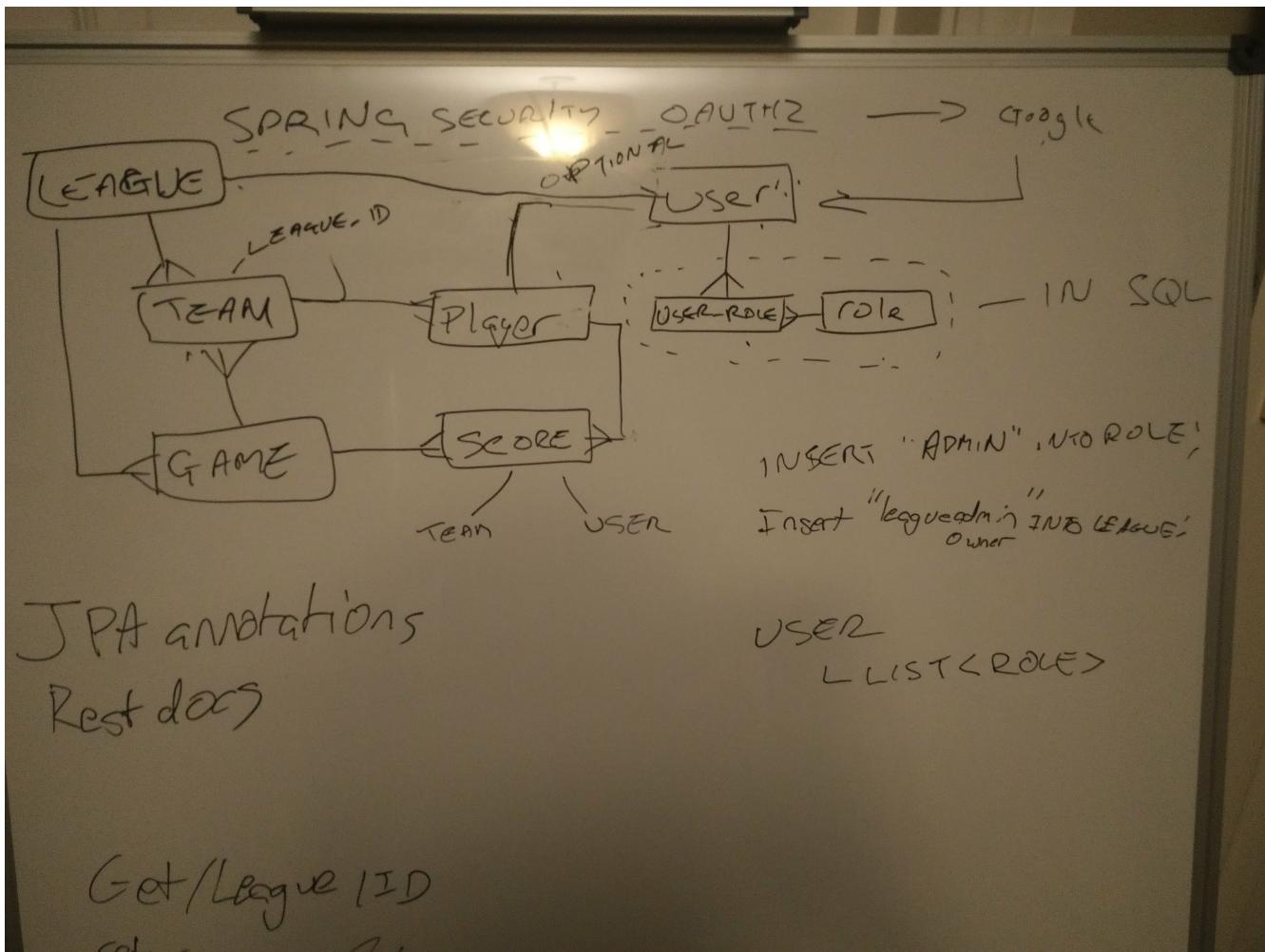
Player{
    name: String
    score: Score Object
}

User{
    ID: 47q047309-47120-97410-298490
    team: Team
    player: Player
    rank: leagueAdmin || teamOwner || scoreAdder
}

Game{
    score: Score Object
    time: yyyy/mm/dd
    venue: Venue
    status: complete || in progress || not started
}

```

2.2. Database modelling



Database modelling: All objects

This was the initial modelling of the database. It shows the relationship between all the objects. User and Role are for security and will be added later than the others.

2.3. Api planning

An API should follow a pattern to be easy to predict where one can make or retrieve an object. I decided to have a base url for the back end of /api/ this allows me to change it and add it as new base url later and easily separate the front and back end usage of the application. Past this it makes sense that /{ObjectName}/{id} should be where one would get, delete or modify an object. Where many will need to be retrieved by a user, a /{ObjectName} should get all or all with certain criteria.

2.4. Technologies

I will use springboot and java for the backend. I will use this as java runs well on many platforms. Springboot makes it easy for me to add things into the project. The front end will be written in React JS. I used this as it is very easy to find documentation and sources on how to write it. I will also use material design for styling. The database structure will probably be best in a relational database so a springboot SQL database will be used such as H2 and JPA to communicate nicely. I will use nightwatch for integration testing. This will allow me to automatically use the website, expect behaviour, and take screenshots during. I will use a python script to put all the testing data into the writeup automatically. To allow easy script modification of the writeup it will be written in

asciidoc and made into a pdf with asciidoctor-pdf, the source is plain text and can be easily modified. It supports code highlighting etc as well.

2.5. Design

Upon talking to the client it was established that all original features of the site were important. There will be support for multiple types of tournament including elimination and round robin brackets.

A logged in user will be defaulted to a page of their and their teams statistics.

The style is not important other than the website should be mobile compatible.

There should be support for multiple leagues.

To start a new season of a league it should duplicate the previous and delete parts. To move players only a leagueAdmin can move them.

A shortcut should be made to use the API without a frontend to allow for scripts to do a task faster.

2.6. Research

I did some research on web applications with data support. Spring boot with react seems to be a popular solution due to the support with both web controllers and also JPA support for sql databases. After modelling the database it appears that a relational database would be most fitting so I used a sql database in H2 as it is easy to setup in <http://start.spring.io/>. I decided to use react despite facebook licensing due to it being a popular language with many examples online. React allows me to write JSX which is similar to HTML.

2.7. Testing

Testing will largely be automated by use of unit testing and nightwatch.

Nightwatch will be used in order to test the entire application by its user interface and report on whether or not it is working as intended. It will also take screenshots of the application to allow a developer to quickly look over these screenshots as opposed to having to navigate the website. This allows for quick testing of all versions without the necessity for user input.

Unit testing will be done to check the functionality of functions. If a function works correctly the test will pass and will then be reported as such. This allows for observation of individual functions to find where errors are occurring.

The data will then be written into the writeup by script in order to have a repeated structure of testing.

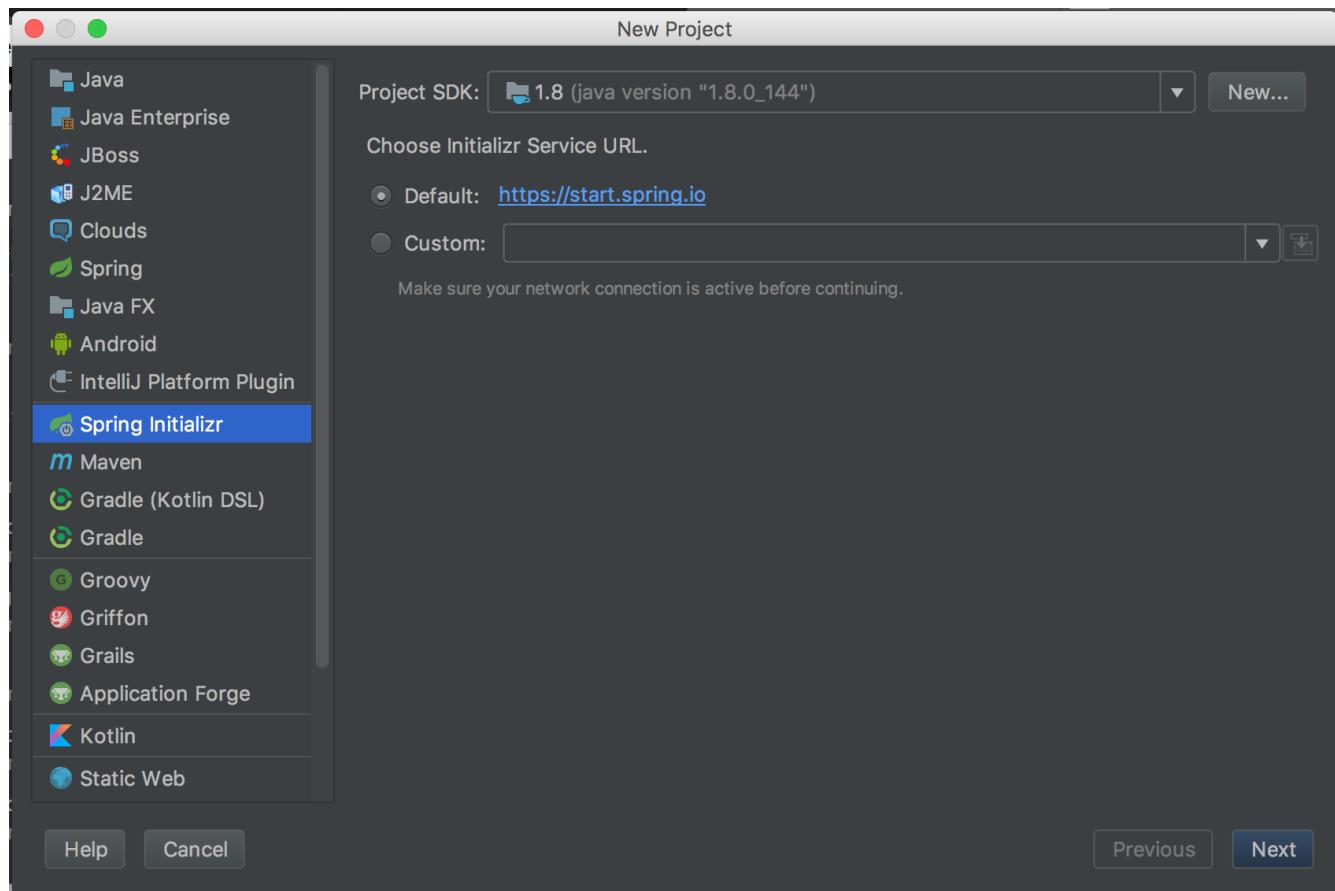
2.8. Ordering

The project will start by laying out the backend for an object, like a league, and then there will be a front end implementation of the object. The ordering will be League, Team, Player, Match, Score then User and roles for security.

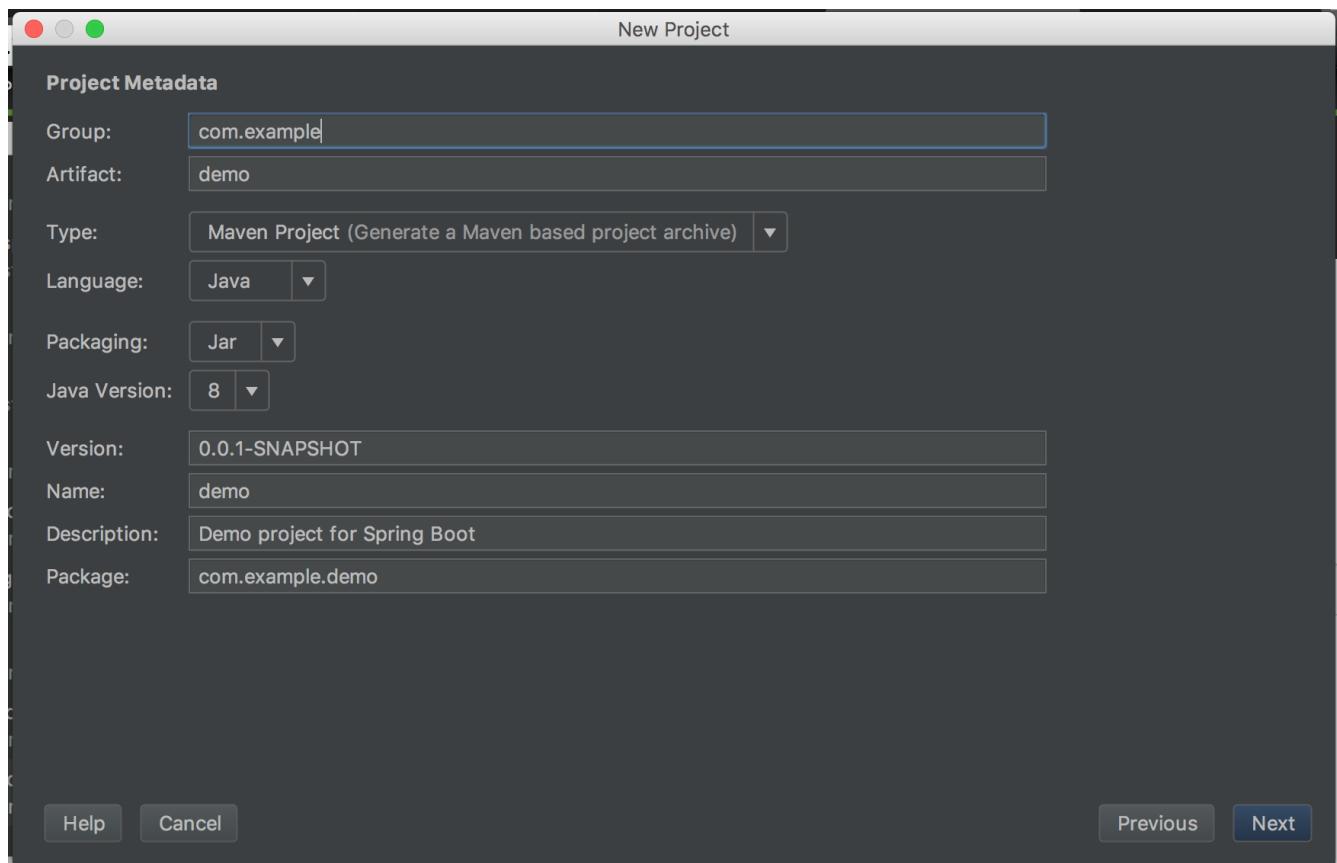
3. Development

3.1. Initial Setup

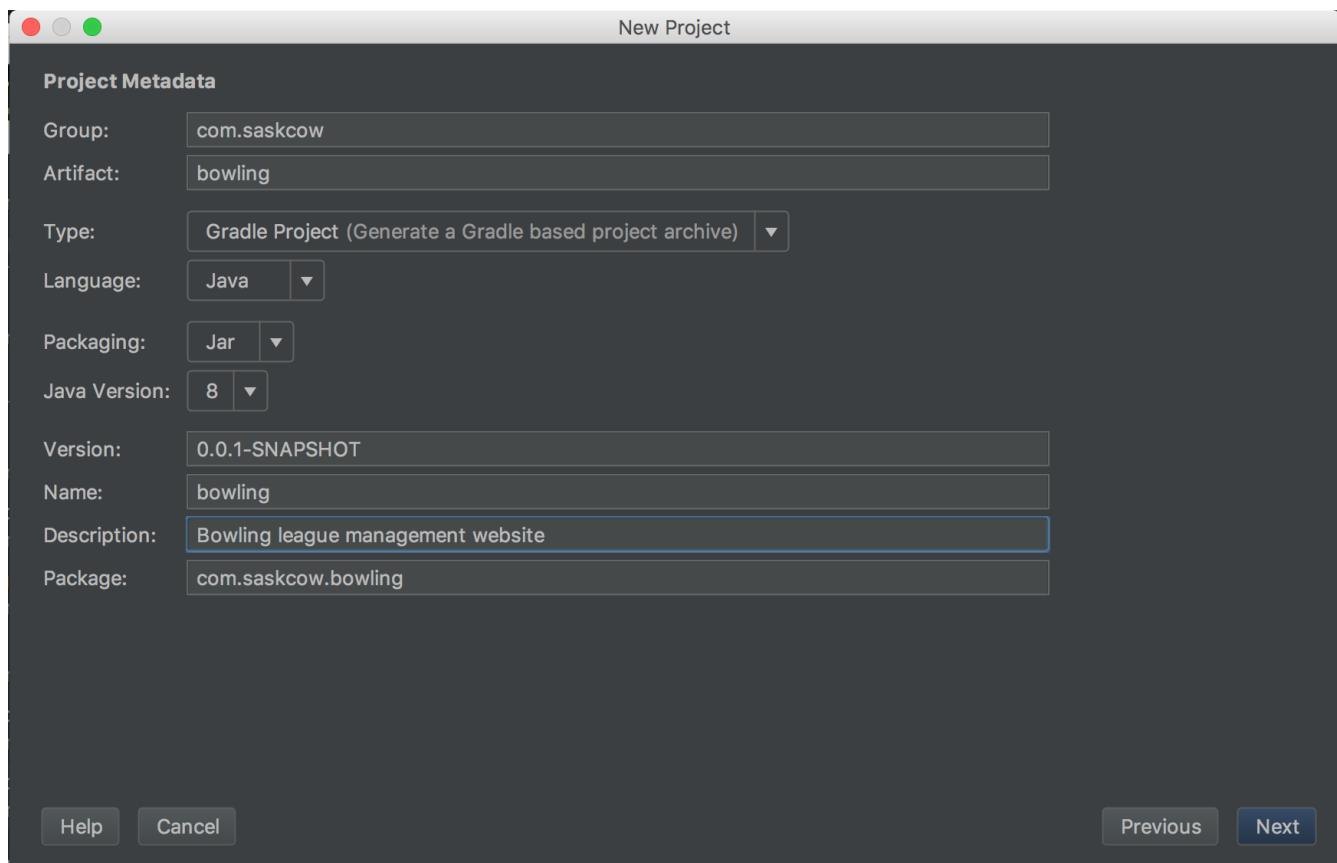
Using IDEA and springboot I can generate a springboot project with certain libraries pre added.



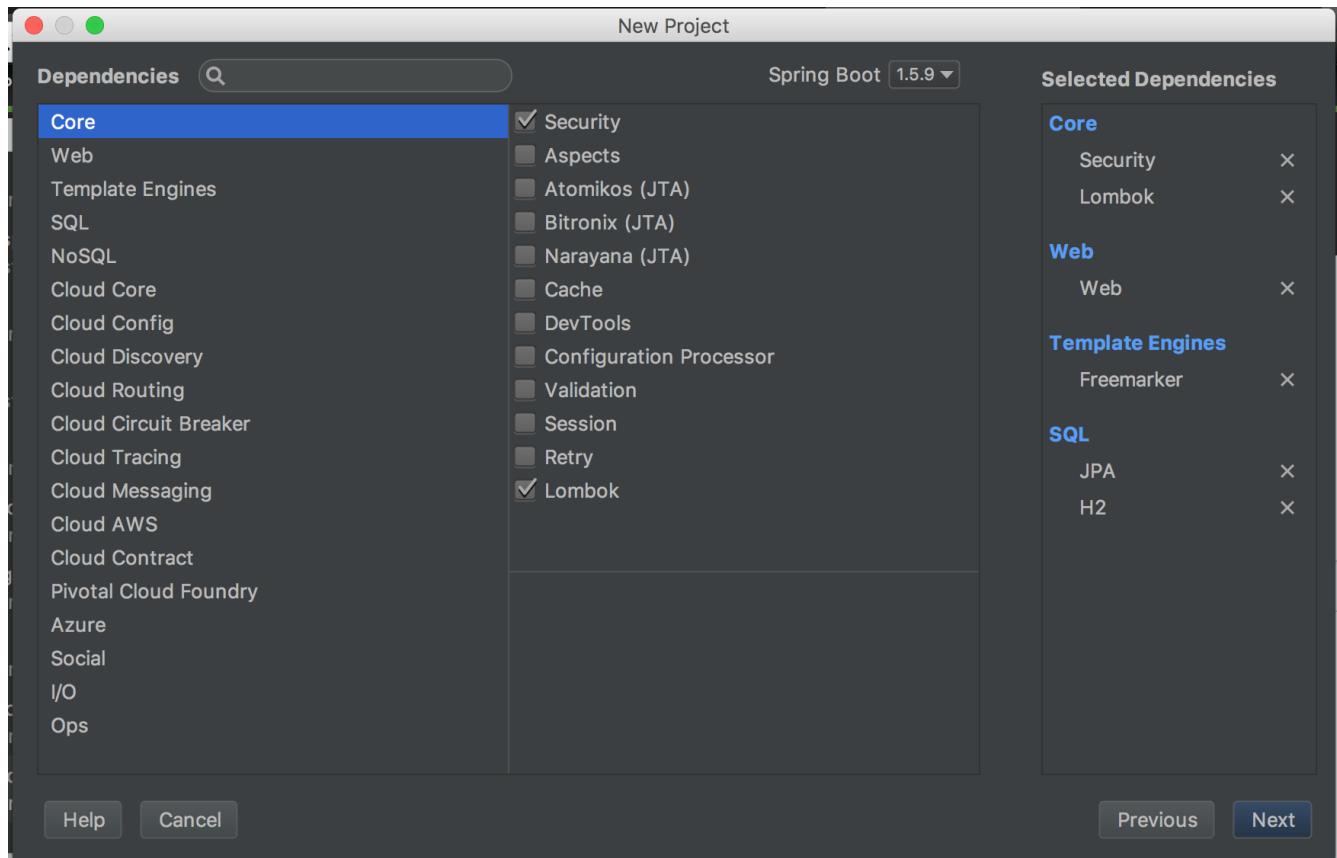
Step 1: New Projects



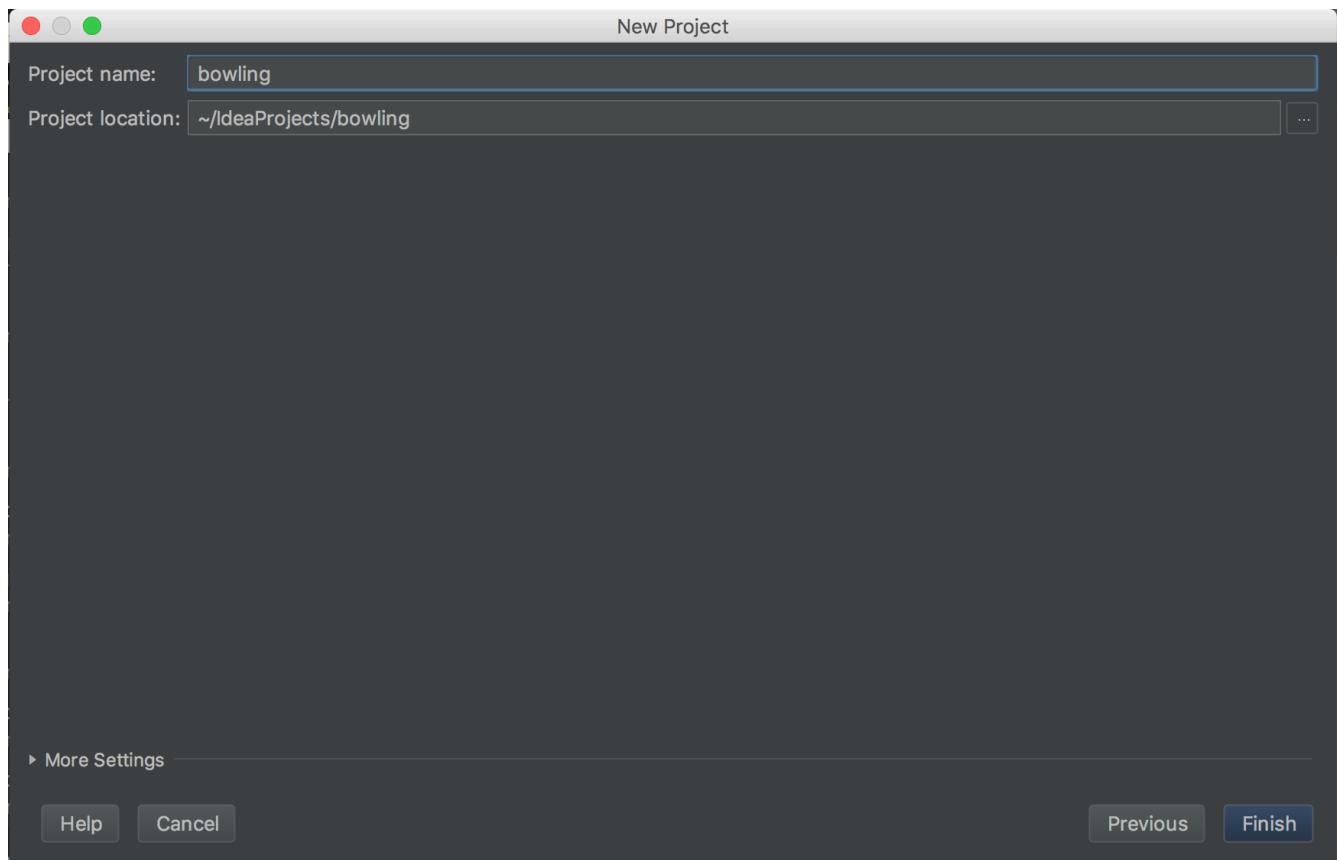
Step 2: Springboot Project



Step 3: Setting up metadata



Step 4: Choosing dependancies



Step 5: Naming Project

This creates us a simple springboot project with a pre-made build.gradle and a BowlingApplication and BowlingApplicationTests.

Theses can be left alone for now

3.2. Modelling The Objects

We already decided what objects should do what, so now we add most of the objects To start not all the objects were added so core functionality could be completed first.

Inside the src/main/java/ folder we go into com.saskcow.bowling, this is where the domain package is made inside this is where the objects will be stored, each object should have most of the features in planning Notably scores and users have been omitted in this version.

3.2.1. League.java

Imports condensed

```
package com.saskcow.bowling.domain;

import ...

@Data // Creates a getter and setter for each property
@Entity // JPA annotation, makes it good to store in a database
@NoArgsConstructor // Creates a constructor with nothing
@AllArgsConstructor // Creates a constructor with everything
public class League {
    private @Id @GeneratedValue Long id;
    // Generate a long value to be used as ID, always unique
    private String name;
    // Name Property of the class
    @OneToMany(mappedBy = "league", cascade = CascadeType.ALL)
    // This prevents a "Failed to load ApplicationContext" error
    // Additionally the properties of it say if the league is deleted, as are all the
    teams
    private List<Team> teams;
    // List<Team> just a list of the teams, type specified in java

    public League(String name, List<Team> teams){
        // A constructor, sets name and teams to what it was created with ID
        autogenerated
        this.name = name;
        this.teams = teams;
    }
}
```



Ensure @ManyToMany etc set to avoid "Failed to load ApplicationContext"

3.2.2. Team.java

Imports condensed

```

package com.saskcow.bowling.domain;

import ...

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
//Same as with League
public class Team {
    private @Id @GeneratedValue Long id;
    private String name;
    @OneToMany(mappedBy = "team", cascade = CascadeType.ALL)
    // Players part of team, so a team will change players on its change
    private List<Player> players;
    @ManyToMany
    // Teams have many games, games have 2 teams so a manytomany is identified
    private List<Game> games;
    @ManyToOne
    private League league;

    // A constructor with everything but generating the id
    public Team(String name, List<Player> players, List<Game> games, League league) {
        this.name = name;
        this.players = players;
        this.games = games;
        this.league = league;
    }

    // A second constructor is put in place in order to create a team which has no
    // players or games
    public Team(String name, League league) {
        this.name = name;
        this.league = league;
        this.players = new LinkedList<>();
        this.games = new LinkedList<>();
    }
}

```

3.2.3. Player.java

Imports condensed

Very similar to previous, nothing new used.

```

package com.saskcow.bowling.domain;

import ...

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
//Same as with League
public class Team {
    private @Id @GeneratedValue Long id;
    private String name;
    @ManyToOne
    private Team team;

    public Player(String name, Team team) {
        this.name = name;
        this.team = team;
    }
}

```

3.2.4. Game.java

Imports condensed

Also very similar to previous, nothing new used.

```

package com.saskcow.bowling.domain;

import ...

@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Game {
    private @Id @GeneratedValue Long id;
    private LocalDateTime time;
    private String venue;
    @ManyToMany
    private List<Team> teams;
    // Should only ever have 2 values, not enforced

    public Game(LocalDateTime time, String venue, List<Team> teams) {
        this.time = time;
        this.venue = venue;
        this.teams = teams;
    }
}

```

3.2.5. Errors at this stage

This stage was fairly simple so few errors occurred other than occasional mistypes picked up by the ide as it went along. It is also hard to find errors at this stage due to nothing happening.

@ManyToOne, @OneToMany, @ManyToMany annotations

Without these annotations in place a java.lang.IllegalStateException is raised

```
Failed to load ApplicationContext
java.lang.IllegalStateException: Failed to load ApplicationContext
    at org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate.loadContext (DefaultCacheAwareContextLoaderDelegate.java:124)
    at org.springframework.test.context.support.DefaultTestContext.getApplicationContext (DefaultTestContext.java:83)
    at org.springframework.test.context.web.ServletTestExecutionListener.setUpRequestContextIfNecessary (ServletTestExecutionListener.java:189)
    at org.springframework.test.context.web.ServletTestExecutionListener.prepareTestInstance (ServletTestExecutionListener.java:131)
    at org.springframework.test.context.TestContextManager.prepareTestInstance (TestContextManager.java:230)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.createTest (SpringJUnit4ClassRunner.java:228)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner$1.runReflectiveCall (SpringJUnit4ClassRunner.java:287) <1 internal call>
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.methodBlock (SpringJUnit4ClassRunner.java:289)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.runChild (SpringJUnit4ClassRunner.java:247)
    at org.springframework.test.context.junit4.SpringJUnit4ClassRunner.runChild (SpringJUnit4ClassRunner.java:94) <5 internal calls>
    at org.springframework.test.context.junit4.statements.RunBeforeTestClassCallbacks.evaluate (RunBeforeTestClassCallbacks.java:61)
    at org.springframework.test.context.junit4.statements.RunAfterTestClassCallbacks.evaluate (RunAfterTestClassCallbacks.java:70) <1 internal call>
```

Figure 1. Start of stacktrace

by adding the

```
@ManyToOne
@OneToMany
@ManyToMany
```

this exception is no longer raised allowing it to compile

3.3. Repositories

Repositories in java are very simple to implement, all repositories can be created with *{ObjectName}* to be substituted with *League, Team, Player and Game*

```
package com.saskcow.bowling.repository;

import ...

// Create a new Repository which copies a CrudRepository so it has all the functions
// <{ObjectName}, Long> shows it stores {ObjectName} by a Long, the Long being the id
// of the object.
public interface {ObjectName}Repository extends CrudRepository<{ObjectName}, Long> { }
```

At this stage this is all that was done for each object and it was saved as *{ObjectName}Repository.java* inside *com.saskcow.bowling.repository*

3.3.1. Errors at this stage

Errors this stage were created by trying to figure out whether or not some CrudRepository things

were worth changing
They weren't.

CrudRepository checking

If the <{ObjectName}, Long> is changed it stores a different type object and returns a different type, so the object cannot be retrieved as itself.

```
C:\Users\Saskcow\IdeaProjects\bowling\src\main\java\com\saskcow\bowling\controller\GameController.java:39: error: incompatible types: String cannot be converted to Game
    Game game = repo.findOne(id);
                           ^

```

Figure 2. Compiler Error

this occurs as a Game object cannot be made from a String, which is the returned object.

3.4. Controllers

There are a lot of controllers in this project, and they are a crucial part to communicate with the frontend to start with, a HomeController will be made to return some basic HTML

3.4.1. index.ftl

This file is where all the front end will be, currently it will just show a blank page, due to spring security, a default password can be set in spring application.properties

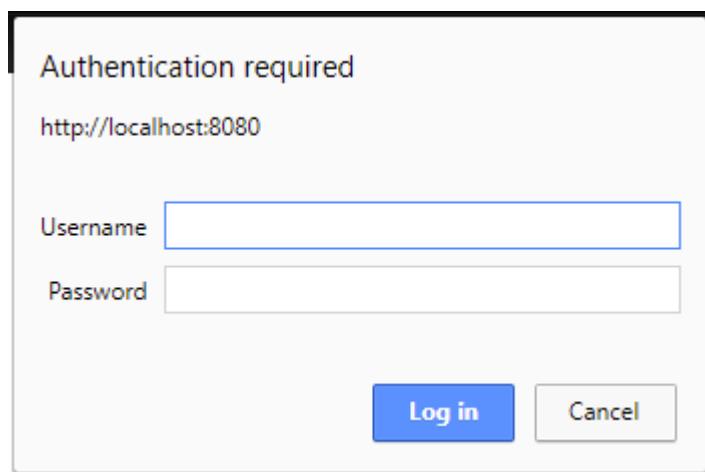


Figure 3. Authentication

```

<html>
<head lang="en">
    <meta charset="UTF-8" />
    <title>Bowling</title>
    <link rel="stylesheet" href="/style.css"/>
    <link rel="stylesheet" href="/material.min.css"/>
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
        <!--These are in place to get some css to make some elements look slightly nicer-->
    <!--/style.css is currently blank-->
</head>
<body>

<div id="react"></div>
<!--create a div for ReactDOM to later use-->

<script src="/built/bundle.js"></script>
<!--load the webpack script, this will be created later-->
<!--webpack is what takes all the js and makes it one file-->

</body>
</html>

```

3.4.2. HomeController.java

This will route to the javascript and ftl to sort out the front end

```

package com.saskcow.bowling.controller;

import ...

@Controller
// Look here for request mappings
public class HomeController {

    @RequestMapping(value = {"/"})
    // Any requests to / call this function
    // Later more will be added as more paths are in the front end
    public String index(){
        return "index";
        // Show index file extension guessed, currently .ftl
    }
}

```

3.4.3. Errors at this stage

Only here as they are waiting for future things, which don't exist yet. Or due to user error.

404 Errors

Due to some parts being missing, the browser shows errors in console where it can't find bundle.js or other parts.

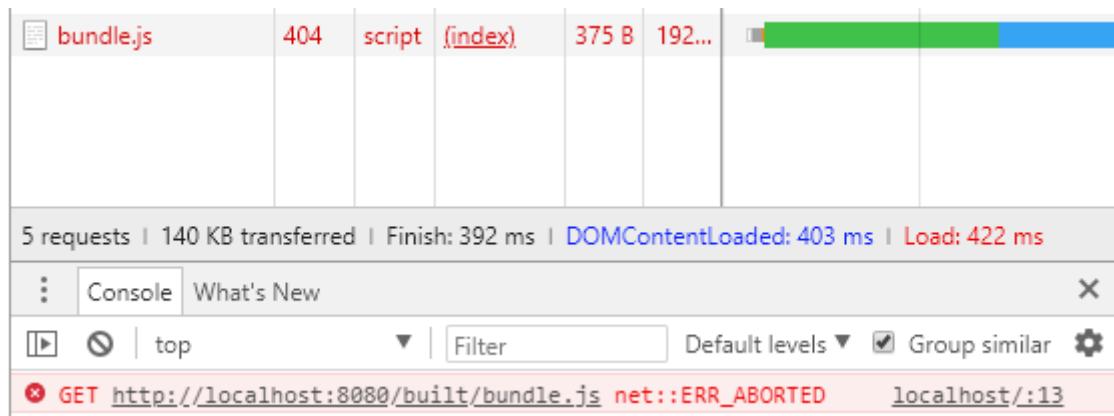


Figure 4. 404 on bundle.js

Multiple Launched Errors

When launched, anything else trying to use the port is blocked, this gives the not especially clear Execution failed for task bootRun as it fails to launch

```
' Exception is:  
org.gradle.api.tasks.TaskExecutionException: Execution failed for task ':bootRun'. <79 internal calls>  
Caused by: org.gradle.process.internal.ExecException: Process 'command 'C:\Program Files\Java\jdk1.8.0_152\bin\java.exe'' finished with non-zero exit value 1 <3 internal calls>  
    at org.springframework.boot.gradle.run.BootRunTask.exec(BootRunTask.java:62) <13 internal calls>  
... 78 more
```

Figure 5. Launch Interrupted

3.5. League Controller

Create mappings for the creating, getting and removing leagues, these will be called from the frontend

3.5.1. LeagueControllerTest.java

Inside the src/test/java I create a folder called controller, this is where I will put controller tests. It is important to test controllers as REST APIs should have consistent features. This helps decide what parts are important, here I identify that we need to be able to make, filter, retrieve and delete leagues as basic functionality.

```
package com.saskcow.bowling.controller;  
  
import ...  
  
@RunWith(MockitoJUnitRunner.class) // This runs the tests with a testRunner, this  
allows assertions which would otherwise be not allowed  
public class LeagueControllerTest {  
  
    @Mock  
    private LeagueRepository repo;
```

```

// Create a LeagueRepository like thing, which does nothing, just pretends it
exists, that's what @Mock does
private MockMvc mockMvc;
// Creates a MockMvc to test the api endpoints

@Before
//Run before tests
public void setUp(){
    mockMvc = MockMvcBuilders.standaloneSetup(new LeagueController(repo)).build();
    // Create a LeagueController and run it
}

@Test
// Run this when running tests, ran by MockitoJUnitRunner
public void addLeague_shouldSaveTheLeague() throws Exception {
    // Doesn't return anything, throws Exception if any part fails, calls test
addLeague_shouldSaveTheLeague
    League league = new League(1L, "Brian", new LinkedList<>());
    // Create an instance of a League, the object from earlier
    Team team = new Team("Team Brian", league);
    league.addTeam(team);
    // Give it a team, because leagues have those
    when(repo.save(isA(League.class))).thenReturn(league);
    // If someone saves a league, return a League, this is what the repo would do,
but the League here is always the same
    when(repo.findOne(league.getId())).thenReturn(league);
    // If someone tries to find this league by its ID, return it
    when(repo.findAll()).thenReturn(Collections.singletonList(league));
    // If someone tries to find all leagues, return this in a list as the only
League

    String uri = mockMvc.perform(post("/api/league")
        // String uri, save the output as a string
        // mockMvc stuff sends a post request to the endpoint
        .content("{\"name\":\"Brian\"}")
        // Send it with this content
        .contentType("application/json"))
        // This content is JSON
        .andExpect(status().isCreated())
        // Should return a 201 (created), if it isn't throw Exception
        .andExpect(header().string("Location",
"http://localhost:8080/api/league/" + league.getId()))
        // Inside the header the location of where the saved object can be
retrieved should be present
        .andReturn().getResponse().getHeader("Location");
        // Save the location header to uri

    mockMvc.perform(get("/api/league"))
        // Send a get request to the endpoint
        .andExpect(status().isOk())
        // Check status is 200 (OK)
}

```

```

        .andExpect(MockMvcResultMatchers.jsonPath("$.name", equalTo("Brian")));
    }
}

// Check that the JSON is an array with size 1
.andExpect(MockMvcResultMatchers.jsonPath("$[0].name", equalTo("Brian")));
// Check the first part of the json has a name of "Brian", like the
league earlier

mockMvc.perform(get(uri))
    // Send a get request to where the location of the league is
    .andExpect(status().isOk())
    .andExpect(MockMvcResultMatchers.jsonPath("$.name", equalTo("Brian")))
    // Check that it has the name Brian
    .andExpect(MockMvcResultMatchers.jsonPath("name", equalTo("Brian")));
    // Check again, by a slightly different method

}

@Test
public void getLeague_shouldFilter() throws Exception {
    League dave = new League(1L, "Dave", null );
    League david = new League(2L, "David", null );
    League brian = new League(3L, "Brian", null);
    // Create 3 leagues
    when(repo.findAll()).thenReturn(Arrays.asList(dave, david, brian));
    // when it calls findByNameContaining("Dav") then it should return all which
    have "Dav" in the name
    // findByNameContaining must be added as it is not in CrudRepository
    when(repo.findByNameContaining("Dav")).thenReturn(Arrays.asList(dave,
david));#
    // Same but with Bri
    when(repo.findByNameContaining("Bri")).thenReturn(Collections.
singletonList(brian));

    // We don't do the post request as one already exists
    mockMvc.perform(get("/api/league?name=Dav"))
        // call the endpoint with a query string with name=Dav
        .andExpect(status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.name", hasSize(2)))
        // Expect 2 items in the returned array
        .andExpect(MockMvcResultMatchers.jsonPath("$[0].name", equalTo(
"Dave")))
        .andExpect(MockMvcResultMatchers.jsonPath("$[1].name", equalTo(
"David")));
    // Expect that the list is as expected, we only know the order as it
    is set earlier
    // Ordinarily order can not be expected

    mockMvc.perform(get("/api/league?name=Bri"))
        // get all leagues with Bri
        .andExpect(status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.name", hasSize(1)))

```

```

        .andExpect(MockMvcResultMatchers.jsonPath("$.name", equalTo(
    "Brian")));
    // Check it has 1 item, which is Brian
}

@Test
public void deleteLeague_shouldDeleteLeague() throws Exception {
    doNothing().when(repo).delete(isA(Long.class));
    // When repo.delete is called with an ID, do nothing, nothing at all

    mockMvc.perform(delete("/api/league/1"))
        // Send a delete request
        .andExpect(status().isNoContent());
    // Expect a 204, No Content is returned
    verify(repo, times(1)).delete(1L);
    // Check it actually called delete
}
}

```

Note at this point there is no controller, so all this fails, and is therefore an error, this is fixed by creating the controller.

```
C:\Users\Saskcow\IdeaProjects\bowling\src\test\java\com\saskcow\bowling\controller\LeagueControllerTest.java:42: error: cannot find symbol
    mockMvc = MockMvcBuilders.standaloneSetup(new LeagueController(repo))
                           ^
  symbol:   class LeagueController
  location: class LeagueControllerTest
1 error
:compileTestJava FAILED
FAILURE: Build failed with an exception.
* What went wrong:
Execution failed for task ':compileTestJava'.
```

Figure 6. No Controller Exists

3.5.2. LeagueController.java

Next is to create the controller, this will route all requests concerning leagues. It will return ResponseEntity s this allows for HTTP codes such as 200 or 204 to be sent, as the tests expected

```

package com.saskcow.bowling.controller;

import ...

@Controller
public class LeagueController {

    private LeagueRepository repo;
    // Has an instance of LeagueRepository

    @Autowired
    // Gets it from application context and injects automatically
    public LeagueController(LeagueRepository repo){
        // Constructor sets repository from app context
        this.repo = repo;
    }
}

```

```

}

@RequestMapping(value = "/api/league", method = RequestMethod.GET)
// For most get requests at /api/league run this function
public ResponseEntity<List<LeagueViewSummary>> findLeagues() {
    // Return a ResponseEntity containing an List of LeagueViewSummary objects
    List<LeagueViewSummary> leagues = StreamSupport.stream(repo.findAll()
    (.spliterator(), false)
        .map(LeagueViewSummary::new).collect(Collectors.toList()));
    // Get the leagues, and stream them after splitting the iterator up
    // Map the stream so each value is replaced by the value of new
    LeagueViewSummary(value)
    // Collect this into a list
    // Have to use StreamSupport see error

    /*
    List<LeagueViewSummary> leagues = new ArrayList<>();
    repo.findAll().
        forEach(league -> leagues.add(new LeagueViewSummary(league)));
    */
    // This was the original implementation but it was replaced with streams for
    cleaner code

    return ResponseEntity.ok(leagues);
    // return OK (200) with body of leagues, which has LeagueViewSummary which can
    be converted to JSON
}

@RequestMapping(value = "/api/league", method = RequestMethod.GET, params =
"name")
// For all get requests at /api/league which have ?name={A name} at the end of the
url
public ResponseEntity<List<LeagueViewSummary>> findLeaguesByName(@RequestParam
("name") String name) {
    List<LeagueViewSummary> leagues = repo.findByNameContaining(name).stream()
        .map(LeagueViewSummary::new).collect(Collectors.toList());
    // repo.findByNameContainint(League) already is a list so has a .stream
already
    return ResponseEntity.ok(leagues);
}

@RequestMapping(value = "/api/league/{id}", method = RequestMethod.GET)
// For all get requests with /id , id is variable
public ResponseEntity<LeagueView> findLeague(@PathVariable("id") Long id) {
    // return a LeagueView in a ResponseEntity, take the path variable id as a
variable
    League league = repo.findOne(id);
    // Get the league from the database
    LeagueView leagueView = new LeagueView(league);
    // Make it a LeagueView
    return ResponseEntity.ok(leagueView);
    // Send it off
}

```

```

    }

    @RequestMapping(value = "/api/league", method = RequestMethod.POST)
    // For all post requests to /api/league
    public ResponseEntity<?> saveLeague(@RequestBody League league) {
        // take request body and convert the JSON into a League
        // This will use the constructor with Name as the name in JSON and teams as
        null as teams cannot be shown in JSON
        if(league.getTeams() == null){
            // If by some miracle someone sends teams in JSON, let them
            league.setTeams(new LinkedList<>());
            // Otherwise use this empty list here and add them after
        }
        League savedLeague = repo.save(league);
        // Save the league to the database
        URI location = ServletUriComponentsBuilder
            // Build a URI
            .fromCurrentRequest().path("/{id}")
            // Get the path to here (/api/league) and add /{id} to it
            .buildAndExpand(savedLeague.getId()).toUri();
            // Set the {id} to the id and make the URI
        return ResponseEntity.created(location).build();
        // Send back a created (201) and with a location of where they could find the
        item
    }

    @RequestMapping(value = "/api/league/{id}", method = RequestMethod.DELETE)
    // All delete requests to /api/league/{id} go here
    public ResponseEntity<Void> deleteLeague(@PathVariable("id") Long id) {
        // Get the ID from the request, and return a ResponseEntity with nothing in it
        (Void)
        try {
            repo.delete(id);
            // Delete the league
            return ResponseEntity.noContent().build();
            // Return a no content (204) for successful deletion
        } catch (EmptyResultDataAccessException e) {
            // If the league was not found, do this instead
            return ResponseEntity.notFound().build();
            // Return a not found error
            // Since it's 4XX it lets the user know it's their fault not mine
        }
    }
}

```



Once again this is part of multiple files implementing the features so once again it won't run as TeamViewSummary and TeamView don't exist, also the repository doesn't have a `findByNameContaining`

3.5.3. LeagueRepository.java

So the easiest bit to fix is there is no `findByNameContaining`, this is necessary otherwise [wrong results](#) This just involves adding

```
List<League> findByNameContaining(String name);
```

to `LeagueRepository.java`, this adds the function `findByNameContaining`. Spring takes this and based off of the name of the method creates the JPQL query similar to

```
SELECT x FROM Leagues x WHERE x.name LIKE ?1
```

Due to the generation of the the table being done by spring, this may not be the exact JPQL, details can be found at <https://docs.spring.io/spring-data/jpa/docs/1.5.1.RELEASE/reference/html/jpa.repositories.html#jpa.query-methods.query-creation>

3.5.4. LeagueView.java

This is an object returned by the controller containing what would be a reasonable amount of information about a League. This is necessary as one-one relationships cause problems with converting to JSON, this results in all Requests

```
Request processing failed; nested exception is org.springframework.http.converter.HttpMessageNotWritableException: Could not write JSON: Infinite recursion (StackOverflowError);
```

Figure 7. League and Team Recursion

as found by the tests, to solve this we make view objects.

```

package com.saskcow.bowling.view;

import ...

@Data
// Getters and Setters and more all done
@NoArgsConstructor
//Can make one with nothing
public class LeagueView {
    private Long id;
    private String name;
    private List<TeamViewSummary> teams;
    // Has a list of summary objects, currently these have nothing so no errors occur,
    but it makes things
    private List<GameView> games;
    // Has a list of GameView for the same reason

    public LeagueView(League league) {
        this.name = league.getName();
        this.id = league.getId();

        this.teams = league.getTeams() == null ? new LinkedList<>() : league.
getTeams().stream().map(TeamViewSummary::new).collect(Collectors.toList());
        this.games = league.getGames() == null ? new LinkedList<>() : league.
getGames().stream().map(GameView::new).collect(Collectors.toList());
        // A lot like the streams earlier, but with a ternary operator, so if teams is
        null, it's now an empty list
        // Also the view objects are blank so nothing happens, but no error occurs
    }
}

```

3.5.5. LeagueViewSummary

Sometimes you only want a bit of information about a league, such as the name of it and the id so you can find out more. This can be easily sent with a ViewSummary, which is a very simple description of a League.

```

package com.saskcow.bowling.view;

import ...

@Data
@NoArgsConstructor
public class LeagueViewSummary {
    // A lot like league, only less, currently does the same as teams and games don't
    have view objects
    private Long id;
    private String name;

    public LeagueViewSummary(League league) {
        this.name = league.getName();
        this.id = league.getId();
    }
}

```

3.5.6. Errors at this stage

findByName not containing

originally I tried using findByName, which only matches full name, because of this when I was expecting there to be 1 leagues returned, there were none as it was not an exact match. This is as bad as windows searching, can't have that.

```

expected:<[1]> but was:<[0]>
Expected : 1
Actual : 0

```

to fix this by adding Containing to the end of findByName it now checks if the string is in the name, this works far better.

NullPointer the first

Probably the first of many, this error occurred when I forgot to mock a function

```

Request processing failed; nested exception is java.lang.NullPointerException
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.NullPointerException <2 internal calls>
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:707) <1 internal call>
    at org.springframework.test.web.servlet.TestDispatcherServlet.service(TestDispatcherServlet.java:65)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:790)
    at org.springframework.mock.web.MockFilterChain$ServletFilterProxy.doFilter(MockFilterChain.java:160)
    at org.springframework.mock.web.MockFilterChain.doFilter(MockFilterChain.java:127)
    at org.springframework.test.web.servlet.MockMvc.perform(MockMvc.java:155)
    at com.saskcow.bowling.controller.LeagueControllerTest.addLeague_shouldSaveTheLeague(LeagueControllerTest.java:58) <9 internal calls>
    at org.springframework.restdocs.JUnitRestDocumentation$1.evaluate(JUnitRestDocumentation.java:63)
    at org.junit.rules.RunRules.evaluate(RunRules.java:20) <9 internal calls>
    at org.mockito.internal.runners.JUnit45AndHigherRunnerImpl.run(JUnit45AndHigherRunnerImpl.java:37)
    at org.mockito.runners.MockitoJUnitRunner.run(MockitoJUnitRunner.java:62) <27 internal calls>
    at java.lang.Thread.run(Thread.java:748)

```

Figure 8. Missed Mock of function

This can be fixed in this instance by going to LeagueController.java:59 (further down stack) and

finding that the `savedLeague.getId()` isn't working as there is no mock, `savedLeague` is null.

This can be fixed with

```
when(repo.save(isA(League.class))).thenReturn(league);
```

or a similar mock dependant on what's missing. This fixes the issue.

StreamSupport necessary

With the streams, lists have a `.stream()` method, iterators don't as it turns out.

```
C:\Users\Saskcow\IdeaProjects\bowling\src\main\java\com\saskcow\bowling\controller\LeagueController.java:32: error: cannot find symbol
    List<LeagueViewSummary> leagues = repo.findAll().stream().map(LeagueViewSummary::new).collect(Collectors.toList());
                                         ^

```

Figure 9. No token `stream()`

To fix this these must be converted into streams differently before `.map()` this can be done with `StreamSupport`, this allows for `Spliterator` to go to streams. By using

```
StreamSupport.stream(repo.findAll().spliterator(), false)
```

it takes the iterator and makes it a spliterator before turning it into a stream, ready to `.map` like the other streams.

3.6. Front End Setup

For a front end it's fairly simple to add nicely, however there are important steps to do first. The main steps required now are: * Webpack * Babel, I'm not writing this in ES5 * NPM setup

3.6.1. Npm Setup

Since the other 2 steps rely on node, I need to setup NPM first. NPM is a package manager which can be used for javascript modules, this makes it easy to manage dependencies and setup other things. By running `npm init` npm becomes easy to setup as it asks step by step for parts, for now it can mostly be left default. Dependencies must now be installed. By doing `npm install --save package-name` or `npm install --save-dev package-name` they get added to `package.json`, which should look something like this.

```
{
  "name": "Bowling",
  "version": "1.0.0",
  "description": "Bowling League Management",
  "dependencies": {
    "react": "^15.3.2",
    "react-dom": "^15.3.2",
    "react-router-dom": "^4.2.2",
    "webpack": "^1.12.2"
  },
  "devDependencies": {
    "babel-core": "^6.18.2",
    "babel-loader": "^6.2.7",
    "babel-plugin-transform-decorators-legacy": "^1.3.4",
    "babel-polyfill": "^6.16.0",
    "babel-preset-env": "^1.6.1",
    "babel-preset-react": "^6.16.0",
    "babel-preset-stage-0": "^6.24.1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/saskcow/bowling.git"
  },
  "author": "Callum Tarttelin",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/saskcow/bowling/issues"
  },
  "homepage": "https://github.com/saskcow/bowling#readme"
}
```

This includes the dependencies I added after.

3.6.2. webpack.config.js

I created this top level file which details how webpack should "bundle" the files before use. Webpack should put all the js into one bundle.js file to be sourced from the template (index.ftl). By editing a basic one I found in docs I made

```

const path = require('path');
// import path

module.exports = {
    // Export this JSON
    entry: './src/main/js/app.js',
    // The main file
    devtool: 'sourcemaps',
    cache: true,
    debug: true,
    output: {
        path: __dirname,
        filename: './src/main/resources/static/built/bundle.js'
        // Put the final file where it is expected
    },
    module: {
        loaders: [
            {
                test: path.join(__dirname, '.'),
                exclude: /(node_modules)/,
                loader: 'babel',
                query: {
                    cacheDirectory: true,
                    presets: ['env', 'react', 'stage-0']
                    // presets say to run babel presets from the array
                }
            }
        ]
    }
};

```

This babels and bundles it all into one file.

3.6.3. Babel

Babel is a transpiler, this takes my javascript I write, and turns it into javascript. The difference is I can write ES6, ES next decorators and react JSX, and it will make it into ES5. This is important as no browser fully supports all these features, so babel translates it so that the browser can use it, and I can write in easier to write languages.

3.6.4. Errors At This Stage

Since this is all follow steps and the webpack is following the structure expected, no errors occurred in this step. Which is of course, unheard of, and also rather convenient.

3.7. Front End for leagues

A bit of back end and preparation later and we are ready to create a front end, this will be rather

ugly for now, but later some css can be thrown at it till it looks less bad, but css is not something I'm good at.

3.7.1. Dependencies

First I need to add some dependencies to my package.json the main ones are axios for REST requests, and material-ui and icons to make it look nicer as well as react-router-dom.

```
"dependencies": {  
  "axios": "^0.17.1",  
  "material-ui": "^1.0.0-beta.31",  
  "material-ui-icons": "^1.0.0-beta.17",  
  "react": "^15.3.2",  
  "react-dom": "^15.3.2",  
  "react-router-dom": "^4.2.2",  
  "rest": "^1.3.1",  
  "webpack": "^1.12.2"  
}
```

3.7.2. app.js

In src/main/js App.js is added. This is the main file which the front end is made off of. This will be used to route the front end paths.

```

import ...

class App extends React.Component {
    // React.Component is extended giving this class state, props etc nicely
    render() {
        // Method called when it is rendered on a screen
        return (
            // JSX returned to make into HTML to display
            <Router>
                // React Router BrowserRouter which creates a router which can have routes
                here
                <div className="App">
                    <Route exact path="/" component={LeagueList}/>
                    // If at path / show LeagueList component
                    <Route exact path="/league" component={LeagueList}/>
                    <Route exact path="/league/:id" component={League}/>
                    <Route path="/add/league" component={AddLeague}/>
                    // Could match multiple but not setup to, as I don't want it to
                    </div>
                </Router>
            )
        }
    }
}

ReactDOM.render(
    // When imported this runs and renders some JSX
    <App/>,
    // Render App from above
    document.getElementById('react')
    // Put it in the 'react' element in the template
);

```

3.7.3. LeagueSummary.js

We need a way to display the summary information, which is pretty much just a name and an ID. Using the id to link to more information we are able to just have a hyperlinked name and we can put many of these in a list

```

import ...

class LeagueSummary extends React.Component {
  render() {
    return (
      <li className={this.props.children.replace(/\s+/g, '-').toLowerCase()}>
        // Create a list element with name same as league, "-" instead of " "
        <Link to={'/league/' + this.props.id.toString()}>
          // Link it to "/league/:id" so that it can go to the more details page
          {this.props.children} // Write the name of the league
        </Link>
        <Delete id={this.props.id} type={'league'} name={this.props.children}/>
        // Create a delete for a league with this ID, onclick delete the league
      </li>
    )
  }
}

export default LeagueSummary;

```

3.7.4. LeagueList.js

Here is where the LeagueSummary will be used, this needs to get all the leagues from the java and display them all.

```

import ...

class LeagueList extends React.Component {
  constructor() { // Run when component created
    super(); // make usable
    this.state = ({status: "Loading"}); // Set state to Loading
    this.updateLeagues = this.updateLeagues.bind(this);
    this.refresh = this.refresh.bind(this);
    // Give functions this
    this.updateLeagues() // Get the leagues
  }

  updateLeagues() {
    axios.get('/api/league')
      .then(response => { // When complete
        this.setState({status: "OK", leagues: response.data})
        // Set status and the leagues from the request
      })
      .catch(error => { // Run this if there is an error
        if (error.response) {
          this.setState({status: "error", err: error.response.data});
        } else if (error.request) {
          this.setState({status: "error", err: "No Response"});
          console.log(error.request);
        }
      })
  }
}

```

```

        } else {
            this.setState({status: "error", err: "Error with Request"});
            console.log('Error', error.message);
        }
        // Display error to user, dependant on error
    );
}

refresh() { // Get leagues again and set as loading
    this.setState({status: "Loading"});
    this.updateLeagues()
}

render() {
    console.log(this.state.status);
    if (this.state.status === "Loading") {
        // If it is loading
        return (
            <div className={"Loading"}>
                <CircularProgress color={"primary"} />
                // Create a loading wheel with primary colour
            </div>
        )
    } else if (this.state.status === "Error") {
        // If there is an error, give a reset button and show that there is an error
        return (
            <div className={"Error"}>
                <h2>Error</h2>
                <Button variant={"raised"} color={"primary"} className={"RefreshButton"} onClick={this.refresh}>Refresh Leagues</Button>
            </div>
        )
    } else {
        // If there is no problem
        return (
            <div className={"Leagues"}>
                <ul>
                    // Create an unordered list
                    {this.state.leagues.map(league => (
                        // For league in this.state.leagues
                        <div key={league.id}> // When mapped it likes having a key, otherwise it gives a warning
                            <LeagueSummary id={league.id}>{league.name}</LeagueSummary>
                            // Create a LeagueSummary with name and id
                        </div>
                    )));
                </ul>
                <Button variant={"raised"} color={"primary"} className={"RefreshButton"} onClick={this.refresh}>Refresh Leagues</Button>
                // Create a refresh button, onClick call refresh function
                <Link to={"/add/league"}><Button className={'add'} variant={"fab"}>

```

```

        color={"primary"}><AddIcon /></Button></Link>
        // Create a hyperlink to /add/league with an add icon as a fab button
    </div>
)
}
}
}

export default LeagueList; // Export LeagueList by default when LeagueList.js imported

```

3.7.5. League.js

This is the page corresponding to /api/league/id, at /league/id there will be a front end representation of the data. This will be accessed from when a user clicks on a link to here.

```

import ...

class League extends React.Component {

constructor(props) { // give access to props in constructor
    super();
    this.state = {status: "Loading"};
    this.getLeague = this.getLeague.bind(this);
    this.getLeague(props.match.params.id); // Call with the id from /:id
}

getLeague(id) {
    axios.get('/api/league/' + id)
        // Get a LeagueView
        .then(response => {
            this.setState({
                status: "OK",
                id: id,
                // Status is done, ID is set in case needed
                name: response.data.name,
                teams: response.data.teams,
                games: response.data.games
                // Set this.state.* to corresponding value from response
            })
        })
        .catch(error => {
            // Error catching
            if (error.response) {
                this.setState({status: "error", err: error.response.data});
            } else if (error.request) {
                this.setState({status: "error", err: "No Response"});
                console.log(error.request);
            } else {
                this.setState({status: "error", err: "Error with Request"});
                console.log('Error', error.message);
            }
        })
}

```

```

        }
    });

render() {
    if(this.state.status === "OK") {
        return (
            <div className={'League'}>

                <h2>{this.state.name}</h2>
                <Link to={"/league"}>Back</Link>
                // Title and link back to previous part

            </div>
        )
    } else if (this.state.status === "error") {
        // If there is an error, show the error
        return (
            <h2>{this.state.err}</h2>
        )
    } else {
        // If loading, loading circle
        return <CircularProgress color={"primary"} />
    }
}

export default League;

```

3.7.6. AddLeague.js

Page where one can add a league.

```

import ...

class AddLeague extends React.Component {
    constructor() {
        super();
        this.state = ({name: "No Name"});
        // Default name is No Name
        this.submit = this.submit.bind(this);
        this.updateName = this.updateName.bind(this);
    }

    submit(event) {
        event.preventDefault();
        axios.post("/api/league", {name: this.state.name})
            // Post what the name of the new league is
            .then(response => {
                window.location.href = '/league';

```

```

    // Redirect to '/league'
    this.state.name = "";
    console.log("created at " + response.headers.location);
    // Reset and log things
  })
  .catch(function (error) {
    console.log(error);
    // Simple error catching
  });
}

updateName(event) {
  this.setState({name: event.target.value})
  // change name in state to value of TextField
}

render() {
  return (
    <div className={"AddScreen"}>
      <h1>Add a league!</h1>
      <form className={"theLeagueForm"} onSubmit={this.submit}> // when submitted
        cakk submit
          <TextField // Create a text entry
            id="LeagueName"
            label="League Name" // Create a nice floating label on edit
            placeholder="League Name"
            className={"LeagueNameInput"}
            onChange={this.updateName} // Update name in state whenever this is
changed
          /> <br />
          <Button type="submit" variant="raised" color="primary"
            className={"submitForm"}>Submit</Button>
          // Submit form
        </form>
      </div>
    )
  }
}

export default AddLeague;

```

3.7.7. Delete.js

It is useful to be able to delete leagues. Sometimes too many are created, or some badly named, it's nice not to have to create " BowlingLeague"); DROP TABLE Leagues; — ", which sadly doesn't even work

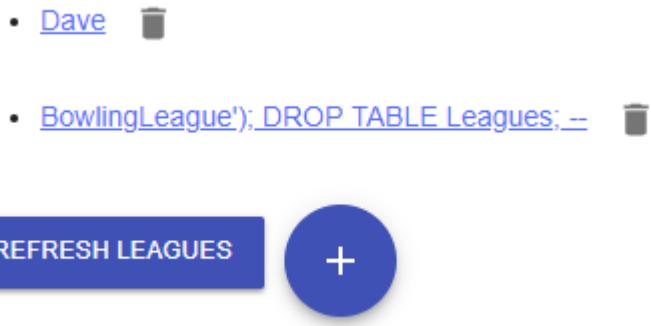


Figure 10. Protected against SQL Injection

So a deletion method is in order, so we need something to call the Delete from the backend.

```

import ...

class LeagueList extends React.Component {
  constructor(props) { // Have props in constructor
    super();
    this.state = ({status: "Loading", id: props.id, type: props.type});
    // Set type (currently league) and the id
    this.deleteThing = this.deleteThing.bind(this);
  }

  deleteThing() {
    axios.delete('/api/' + this.state.type + '/' + this.state.id)
      // Call at the delete url for the id and type
      .then(response => {
        location.reload(); // Refresh page
        console.log("deleted " + response.headers.location); // Log deletion
      })
      .catch(error => {
        // error catching
        if (error.response) {
          this.setState({status: "error", err: error.response.data});
        } else if (error.request) {
          this.setState({status: "error", err: "No Response"});
          console.log(error.request);
        } else {
          this.setState({status: "error", err: "Error with Request"});
          console.log('Error', error.message);
        }
      });
  }

  render() {
    if (this.state.status === "Error") {
      return (
        <h2>{this.state.err}</h2> // If error, show error
      )
    } else {
      return (
        // Create a button, which looks like a delete icon, which when clicked, sends
        // the delete request
        <IconButton onClick={this.deleteThing} id={"delete-" +
        this.props.id.toString()} name={"delete-"+this.props.name.replace(/\s+/g, '-').
        toLowerCase()}><DeleteIcon /></IconButton>
      )
    }
  }
}

export default LeagueList;

```

3.7.8. Errors at this stage

There were numerous errors at this stage, some harder to fix than others.

Enter to submit

When using the add league form, the form, could not be submitted by pressing enter. This can be fixed by instead of button.

```
+ <Button type={"submit"} variant={"raised"} color={"primary"} className={"submitForm"}>Submit</Button>  
  
+ <form className={"theLeagueForm"} onSubmit={this.submit}>  
  
- <form className={"theLeagueForm"}>  
  
- <Button onClick={this.submit} variant={"raised"} color={"primary"} className={"submitForm"}>Submit</Button>
```

This now means that the form is additionally submitted with enter.

Second time lucky redirects

There was a really peculiar error, causing this in the testing.

2 tests, 1 failed, 1 errors,

Classname	Name	Time	Assertions
TestLeague	Test adding leagues	7.204	3
Testing if element <.Leagues> contains text: "nightwatch". Element could not be located.			
None	None	None	None
TestLeague	Test removing leagues	None	None
TestLeague		7.204	

To fix this I tried to find an example from react docs and compare. I noticed that there was an `event.preventDefault()`, so this was added to the start of `submit` and it stopped happening. It appears that the default behaviour in some way blocked the first redirect.

Button imports

Inputs are condensed in the writeup, for the buttons I am using material-ui, because the buttons look nice, and I intend to eventually fit the rest of the page to material design. `import {Button, TextField} from 'material-ui';` is the import in the js. But before it was `import Button from 'material-ui';` this was importing the entirety of material-ui and calling it Button. This was not a valid component and caused

```
warning.js:33 Warning: React.createElement: type is invalid -- expected a string (for built-in components) or a class/function (for composite components) but got: undefined. You likely forgot to export your component from the file it's defined in. Check the render method of 'AddLeague'. +
  in AddLeague (created by Route) +
  in Route (created by App) +
  in div (created by App) +
  in Router (created by BrowserRouter) +
  in BrowserRouter (created by App) +
  in App
```

Calling incorrect API path

Sometimes typographical errors can be annoying, I had a request towards the wrong path, this caused permanent loading and also caused a 404 error on the get request, it also complained due to content being empty it cannot map undefined.

This was easily fixed by changing `/api/legue/` to `/api/league`

Material-ui changes

When using early versions of libraries, the components are sometimes subject to change. On a new version of material-ui buttons changed, this cause buttons to look different and log an error saying Warning: Unknown prop `raised` on `<button>` tag. Remove this



Figure 11. Image of button

This was fixed by `<Button raised /> ⇒ <Button variant="raised">`.

404 on paths

Home controller didn't have all the routes, so when not linked through js, the backend would say it doesn't exist and show.

Whitelabel Error Page

This application has no explicit mapping for `/error`, so you are seeing this as a fallback.

Tue Feb 27 19:50:56 GMT 2018

There was an unexpected error (type=Not Found, status=404).

No message available

Figure 12. 404 error

```

@RequestMapping(value = {"/league/*/add-team", "/league/*/add-game", "league/*", "/",
    "/add/league", "/league", "/player/*", "/team/*", "/team/*/add-player"})
public String index(){
    return "index";
}

```

This added all current and soon to be added routes, so that this would no longer be a problem.

3.8. Team Controller

Adding teams was fairly similar to adding leagues.

3.8.1. TeamControllerTest.java

This is very similar to LeagueControllerTest as both do similar things.

```

package com.saskcow.bowling.controller;

import ...

@RunWith(MockitoJUnitRunner.class)
// This runs the tests with a testRunner, this allows assertions which would otherwise
be not allowed
public class TeamControllerTest {

    @Mock
    private TeamRepository repo;
    // Create a mocked TeamRepository, doesn't do anything which I don't tell it to do
    // explicitly
    @Mock
    private LeagueRepository leagueRepository;
    // Create a mocked leagueRepository
    private MockMvc mockMvc;
    // Needs a MockMvc to send requests to test the app

    @Before
    public void setUp() {
        mockMvc = MockMvcBuilders.standaloneSetup(new TeamController(repo,
leagueRepository)).build();
        // Create the TeamController
    }

    @Test
    public void addTeam_shouldSaveTheTeam() throws Exception {
        League league = new League(1L, "Brian", new LinkedList<>());
        // Create a league
        Team team = new Team(1L, "Brian", null, null, league);
        // Add a team to the league
    }
}

```

```

when(repo.save(isA(Team.class))).thenReturn(team);
when(repo.findOne(team.getId())).thenReturn(team);
// When I save, or try to find the team, return the team
when(repo.findAll()).thenReturn(Collections.singletonList(team));
// When I try to find all teams, return a list of teams
when(leagueRepository.findOne(league.getId())).thenReturn(league);
// When I try to find the league, return the league

String uri = mockMvc.perform(post("/api/team")
    // Send a post request to /api/team
    .content("{\"name\":\"Brian\", \"leagueId\": \"1\"}")
    // Send this
    .contentType("application/json")
    // With contentType: 'application/json' header
    .andExpect(status().isCreated())
    // Expect a 201
    .andExpect(header().string("Location",
"http://localhost:8080/api/team/" + team.getId()))
    // Expect a header called location with the location of the team
    .andReturn().getResponse().getHeader("Location");
// Return the location

mockMvc.perform(get(uri))
    // Send a get request to location returned from post
    .andExpect(status().isOk())
    // Expect 200
    .andExpect(MockMvcResultMatchers.jsonPath("$.name", equalTo("Brian")))
    .andExpect(MockMvcResultMatchers.jsonPath("name", equalTo("Brian")));
    // Expect it has the name Brian, like the team
}

@Test
public void deleteTeam_shouldDeleteTeam() throws Exception {
    League league = new League(1L, "Brian", new LinkedList<>());
    Team team = new Team(1L, "Brian", null, null, league);
    Team team2 = new Team(2L, "Dave", null, null, league);
    league.addTeam(team);
    league.addTeam(team2);

    // Create the objects again

    doNothing().when(repo).delete(isA(Long.class));
    // When I delete something, do nothing
    when(repo.findOne(team.getId())).thenReturn(team);
    // When I try to find a team, return the team

    mockMvc.perform(delete("/api/team/1"))
        // Send a delete request to the team
        .andExpect(status().isNoContent());
        // Expect a 204 returned
}

```

```

        verify(repo, times(1)).delete(1L);
        // Check that delete attempted
        assertThat(league.getTeams()).doesNotContain(team);
        assertThat(league.getTeams()).contains(team2);
        // Check it removed itself, but not the other team from league
    }
}

```

3.8.2. TeamRest.java

We can't just send a full Team, sadly relational databases are complicated like that. So we use a simpler class we can use to construct a Team.

```

package com.saskcow.bowling.rest;

import ...

@Data
// Getters and Setters and more all done
@NoArgsConstructor
// Can make one with nothing
@AllArgsConstructor
// Can make one with everything
public class TeamRest {
    private String name;
    private Long leagueId;
}

```

This we can construct with the following JSON from the front end.

```
{
    name: "Team Name",
    leagueId: 314
}
```

This makes it easy to construct from the JSON if we say that what we are sent is this.

3.8.3. TeamController.java

The controller which maps /api/team requests as the tests expected.

```

package com.saskcow.bowling.controller;

import ...

import java.net.URI;

```

```

@Controller
public class TeamController {

    private TeamRepository repo;
    private LeagueRepository leagueRepository;
    // Needs a TeamRepository and a LeagueRepository

    @Autowired
    // Gets it from application context and injects automatically
    public TeamController(TeamRepository repo, LeagueRepository leagueRepository){
        this.repo = repo;
        this.leagueRepository = leagueRepository;
    }
    // Create a TeamController with stuff from application context

    @RequestMapping(value = "/api/team/{id}", method = RequestMethod.GET)
    // Any get requests to /api/team/id calls this function
    public ResponseEntity<TeamView> findLeague(@PathVariable("id") Long id) {
        Team team = repo.findOne(id);
        // Get the Team
        TeamView teamView = new TeamView(team);
        // Make it a TeamView
        return ResponseEntity.ok(teamView);
        // Send it back with a 200
    }

    @RequestMapping(value = "/api/team", method = RequestMethod.POST)
    // Any post requests to /api/team call this function
    public ResponseEntity<?> saveLeague(@RequestBody TeamRest team) {
        // Take the JSON payload as a TeamRest called team
        League league = leagueRepository.findOne(team.getLeagueId());
        // Get the league
        Team savedTeam = repo.save(new Team(team.getName(), league));
        // Save a new Team with name of JSON and league of what we just got
        league.addTeam(savedTeam);
        // Add Team to the League
        leagueRepository.save(league);
        // Save League changes
        URI location = ServletUriComponentsBuilder
            .fromCurrentRequest().path("/{id}")
            .buildAndExpand(savedTeam.getId()).toUri();
        // Create a URI with the location
        return ResponseEntity.created(location).build();
        // Return location header with a 201
    }

    @RequestMapping(value = "/api/team/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<Void> deleteLeague(@PathVariable("id") Long id) {
        try {
            Team team = repo.findOne(id);

```

```

        // Get the team
        team.getLeague().deleteTeam(team);
        // Remove team from league
        repo.delete(id);
        // Delete the team
        return ResponseEntity.noContent().build();
        // Return 204
    } catch (ResourceNotFoundException e) {
        return ResponseEntity.notFound().build();
        // If not found, send a 404
    }
}
}

```

3.8.4. TeamView.java

In the same way it's hard to send relational objects as JSON due to infinite recursion, we need a simpler way of sending it.

```

package com.saskcow.bowling.view;

import ...

@Data
@NoArgsConstructor
// We've seen these enough now
public class TeamView {
    private LeagueViewSummary league;
    // Get a simplified league
    private Long id;
    private String name;
    // Get the team id and name
    private List<PlayerViewSummary> players;
    private List<GameView> games;
    // Get players and games as View objects

    public TeamView(Team team) {
        this.league = new LeagueViewSummary(team.getLeague());
        this.name = team.getName();
        this.id = team.getId();
        this.players = team.getPlayers() == null ? null : team.getPlayers().stream()
            .map(PlayerViewSummary::new).collect(Collectors.toList());
        this.games = team.getGames() == null ? null : team.getGames().stream()
            .map(GameView::new).collect(Collectors.toList());
        // Make all games into GameView and all teams into TeamViewSummary
    }
    // Get all the values from a team it was created with
}

```

Makes it into JSON like

```
{  
    "league": {  
        "id": 1,  
        "name": "City Watch"  
    },  
    "id": 3,  
    "name": "The Night Watch",  
    "players": [  
        {  
            "id": 5,  
            "name": "Sam Vimes"  
        },  
        {  
            "id": 6,  
            "name": "Carrot Ironfoundersson"  
        },  
        {  
            "id": 7,  
            "name": "Nobby Nobbs"  
        },  
        {  
            "id": 8,  
            "name": "Fred Colon"  
        }  
    "games": [  
        {  
            "id": 13,  
            "time": "2018-04-15T15:54:53",  
            "teams": [  
                {  
                    "id": 3,  
                    "name": "The Night Watch"  
                },  
                {  
                    "id": 4,  
                    "name": "Cable Street Particulars"  
                }  
            "venue": "The Shades",  
            "league": {  
                "id": 1,  
                "name": "City Watch"  
            }  
}
```



JSON from a later version with all the view objects

3.8.5. TeamViewSummary.java

A simplified teamView for when other objects don't care about the details

```
@Data  
@NoArgsConstructor  
public class TeamViewSummary {  
    private Long id;  
    private String name;  
    // Only needs id and name  
  
    public TeamViewSummary(Team team) {  
        this.name = team.getName();  
        this.id = team.getId();  
    }  
    // Get values from the team  
}
```

Nice and simple, sends like

```
{  
    "id": 3,  
    "name": "The Night Watch"  
}
```

3.8.6. Errors at this stage

This was fairly similar to the previous league controller creation, so few errors occurred.

@RunWith importance

Upon writing tests, I forgot to add the @RunWith annotation. This caused the following,

```
You cannot use argument matchers outside of verification or stubbing.  
Examples of correct usage of argument matchers:  
    when(mock.get(anyInt())).thenReturn(null);  
    doThrow(new RuntimeException()).when(mock).someVoidMethod(anyObject());  
    verify(mock).someMethod(contains("foo"))  
  
This message may appear after an NullPointerException if the last matcher is returning an object  
like any() but the stubbed method signature expect a primitive argument, in this case,  
use primitive alternatives.  
    when(mock.get(any())); // bad use, will raise NPE  
    when(mock.get(anyInt())); // correct usage use
```

Figure 13. No @RunWith annotation

This was easily rectified by adding the annotation

Need View Objects

Now that leagues have teams, and teams have leagues, sending back the league or team causes a stack overflow as both contain references to the other, therefore we require the view objects.

3.9. Team front end

Now the back end for teams is done, we need to give a league some teams, and be able to add and display more detail on these teams.

3.9.1. League.js

Leagues now have teams, let's show that.

```
<h3>Teams</h3>
<ul className={"Teams"}>
  // Create a list of all the teams
  {this.state.teams.map(team => (
    <TeamSummary key={team.id} id={team.id}>{team.name}</TeamSummary> // There are no
    teams, so this is just blank
  ))}
</ul>
<Link to={"/league/" + this.state.id + '/add-team'}><Button className={"addTeam"}>
  variant={"raised"} color={"primary"}>Add A Team</Button></Link>
// Link to the add a team to the league page
```

This allows the league to show a list of its teams.

3.9.2. App.js

We need routes to add and show teams.

```
<Router>
  // React Router BrowserRouter which creates a router which can have routes here
  <div className="App">
    // Following added
    <Route exact path="/team/:id" component={Team}/>
      // at /team/{id} show Team page
    <Route path={"/league/:id/add-team"} component={AddTeam}/>
      // at /league/{id}/add-team show the add team to league page
  </div>
</Router>
```

3.9.3. AddTeam.js

Page to add teams to league id specified in URL

```

import ...

// Create component AddTeam
class AddTeam extends React.Component {
  constructor(props) {
    super();
    // calling super makes this work, as in the keyword this, like on the next line
    this.state = ({name: "No Name", leagueId: props.match.params.id});
    // Set default state
    this.submit = this.submit.bind(this);
    this.updateName = this.updateName.bind(this);
    // Bind this to these functions, allows them to access this, again the keyword
  }

  submit(event) {
    // Take an event, like an onClick or onChange on a Component
    event.preventDefault();
    // Stops normal way of doing it, so it can be overridden by doing this
    axios.post("/api/team", {name: this.state.name, leagueId: this.state.leagueId})
      // Send a post request with the JSON based off of state
      .then(response => {
        window.location.href = '/league/' + this.state.leagueId;
        this.state.name = "";
        console.log("created at " + response.headers.location);
        // Redirect to /league/id and reset variables in case redirect has problems
      })
      .catch(function (error) {
        console.log(error);
        // If something goes wrong, log it in developer console (f12 on most browsers)
      });
  }

  updateName(event) {
    this.setState({name: event.target.value})
    // Take an event and set name to the value of the Component which sent the event
  }

  render() {
    // when shown on screen
    return (
      <div className={"AddScreen"}>
        // Contain in a div, as it has to return one Component
        <h1>Add a Team to the League!</h1>
        <form className={"theTeamForm"} onSubmit={this.submit}>
          // When form submitted call submit with event
          <TextField
            id="TeamName"
            label="Team Name"
            placeholder="Team Name"
            className={"TeamNameInput"}
            onChange={this.updateName}

```

```

        /> <br />
        // Create a text field with labels and when changed calls updateName
        <Button type={"submit"} variant={"raised"} color={"primary"}>
      className={"submitForm"}>Submit</Button>
        // Form submit button
      </form>
    </div>
  )
}
}

export default AddTeam // Export AddTeam for easy imports;

```

3.9.4. TeamSummary.js

Simple version to show in the lists of other objects like League.js

```

import ...

class TeamSummary extends React.Component {
  render() {
    // When displayed
    return (
      <li className={this.props.children.replace(/\s+/g, '-').toLowerCase()}>
        // Create a list element with a class all lower cases with " " => "-"
        <Link to={'/team/' + this.props.id.toString()}>
          // Link to the team page
          {this.props.children}
          // Team name
        </Link>
        <Delete id={this.props.id} type={'team'} name={this.props.children}/>
        // Use the delete object from earlier
      </li>
    )
  }
}

export default TeamSummary // Export for easy imports;

```

3.9.5. Team.js

A page to see details of the team, currently not much

```

import ...

class Team extends React.Component {

  constructor(props) {

```

```

super();
this.state = {status: "Loading"};
this.getTeam = this.getLeague.bind(this);
this.getTeam(props.match.params.id);
// Start by getting team details
}

getTeam(id) {
  axios.get('/api/team/' + id)
  // Send a get request to /api/team/ + the id in the page url
  .then(response => {
    this.setState({
      status: "OK",
      id: id,
      name: response.data.name,
      league: response.data.league,
      players: response.data.players
    })
    // Map response to state
  })
  .catch(error => {
    this.setState({status: "error", err: "problem getting team"})
    console.log(error)
  });
}

render() {
  if(this.state.status === "OK") {
    return (
      <div className='Team'>
        // Contain in Team div
        <h2>{this.state.name}</h2>
        <Link to={"/league/" + this.state.league.id}>{this.state.league.name}</Link>
        // Link back to league
      </div>
    )
  } else if (this.state.status === "error") {
    return (
      <h2>{this.state.err}</h2>
    )
    // Display any error in request
  } else {
    return <CircularProgress color="primary" />
  } // Loading icon
}

export default Team;

```

3.9.6. Errors at this stage

Error logging

Ironically the error on request logging originally tried reading error values such as `error.response.status`, however these were sometimes null. This caused null pointer exceptions when trying to handle errors.

This was fixed by simply stating an error has occurred and doing `console.log(error)`.

3.10. Nightwatch Testing

Well, using a webpage takes time, time which I could be spending watching something else use my website, therefore to save me from the pain of actually doing work, I did this.

3.10.1. nightwatch.json

This is basically a configuration file for nightwatch, I got the default off of the internet and added what I needed.

```
{
  "src_folders" : ["src/test/js"],
  "output_folder" : "reports/data",
  "custom_commands_path" : "",
  "custom_assertions_path" : "",
  "page_objects_path" : "",
  "globals_path" : "",

  "selenium" : {
    "start_process" : true,
    "server_path" : "./node_modules/selenium-standalone/.selenium/selenium-
server/3.9.1-server.jar",
    "log_path" : "",
    "port" : 4444,
    "cli_args" : {
      "webdriver.chrome.driver" : "./node_modules/selenium-
standalone/.selenium/chromedriver/2.35-x64-chromedriver"
    }
  },
  "test_settings" : {
    "default" : {
      "launch_url" : "http://localhost",
      "selenium_port" : 4444,
      "selenium_host" : "localhost",
      "silent": true,
      "desiredCapabilities": {
        "browserName": "chrome"
      },
      "screenshots" : {
        "enabled" : true,
        "path" : "./reports/screenshots/errors",
        "on_failure": true
      }
    }
  }
}
```

src_folders is where the tests are, output is where the logs go.
 test_settings mostly say take screenshots and use chrome.

3.10.2. TestLeague.js

Now we have the immense functionality of adding some leagues, and then give them some teams, we now need to test, just the league bit here.

```
module.exports = {
  // Export this
  'Test adding leagues' : function (browser) {
```

```

    // Create a test called Test adding leagues and it's a function which takes a
    browser
    let directory = "./reports/screenshots/TestLeague/";
    // Set image directory
    browser
        .url('http://user:saskcow@localhost:8080/league')
        // Set url to /league on page, authenticate with basic auth if necessary
        .pause(1000)
        // Wait 1000ms for it to load
        .waitForElementVisible('body', 1000)
        // Check if <body> turns up in the next 1000 ms
        .saveScreenshot(directory + "1-start.png")
        // Save a screenshot of the page in the image directory
        .click('button[class~=add]')
        // Click the <button> which has class containing add
        // This is a CSS selector and is what one would use in CSS to set styles for
        elements
        .waitForElementVisible('input[id=LeagueName]', 1000)
        // Wait 1000ms for an input for LeagueName to be visible
        .setValue('input[id=LeagueName]', 'nightwatch')
        // Write nightwatch in the input box
        .saveScreenshot(directory + "2-adding a league.png")
        // Save a screenshot
        .click('button[class~=submitForm]')
        // Click the submit button
        .pause(1000)
        // Wait for page to load
        .assert.containsText('.Leagues', 'nightwatch')
        // Make sure .Leagues (component with class Leagues) contains the text
        nightwatch
        // This will confirm that the team is there
        .saveScreenshot(directory + "3-Shows League.png")
        // Save another screenshot
        .click('button[class~=add]')
        // Click add button again
        .waitForElementVisible('input[id=LeagueName]', 1000)
        .setValue('input[id=LeagueName]', 'daywatch')
        .click('button[class~=submitForm]')
        .pause(1000)
        .assert.containsText('.Leagues', 'daywatch')
        .saveScreenshot(directory + "4-Second League.png")
        // Repeat the add and check processes
    },
    'Test removing leagues' : function (browser) {
        // Create a new testCase which deletes leagues
        let directory = "./reports/screenshots/TestLeague/";
        browser
            .click('button[name=delete-daywatch]')
            .pause(1000);

```

```

// Click the delete daywatch button
browser.expect.element('.Leagues').text.to.not.contain('daywatch');
browser.expect.element('.Leagues').text.to.contain('nightwatch');
// Expect .Leagues to still have nightwatch, but not daywatch
browser
  .saveScreenshot(directory + "5-Deleted daywatch.png")
  .click('button[name=delete-nightwatch]')
  .pause(1000);
// Delete nightwatch anyway, after of course another screenshot
browser.expect.element('.Leagues').text.to.not.contain('nightwatch');
// Check nightwatch isn't there
browser
  .end();
// Close browser
}
};


```

See, far easier than doing it manually 100 odd times, checks I haven't broken anything too bad

3.10.3. TestTeams.js

Well Teams exist, so those should be tested, so this is the test.

```

module.exports = {
  // Export this
  'Setup' : function (browser) {
    // Create testCase Setup
    let directory = "./reports/screenshots/TestTeams/";
    browser
      .url('http://user:saskcow@localhost:8080/league')
      .pause(1000)
      .waitForElementVisible('body', 1000)
      .click('button[class~=add]')
      .waitForElementVisible('input[id=LeagueName]', 1000)
      .setValue('input[id=LeagueName]', 'Nights Watch')
      .click('button[class~=submitForm]')
      .pause(1000)
      .assert.containsText('.Leagues', 'Nights Watch')
      .saveScreenshot(directory + "1-init league.png")
    // Follow procedure to create a team from last time, just less screenshots this
    time
  },
  'Test Adding Teams' : function (browser) {
    let directory = "./reports/screenshots/TestTeams/";
    browser
      .click('li[class=nights-watch]>a')
    // Click on the <a> inside the <li> with Nights Watch
      .pause(1000)
    // Pause for effect, and load times
  }
};


```

```

    .assert.containsText('h2', 'Nights Watch')
    // Check the header says Nights Watch
    .saveScreenshot(directory + "2-League view.png")
    // Everyone wants to see this empty team, so we screenshot it

    .click('button[class~=addTeam]')
    // Click add Team button
    .waitForElementVisible('input[id=TeamName]', 1000)
    // Wait till input visible
    .setValue('input[id=TeamName]', 'Sam Vimes')
    // Enter Name into box, ignore that Sam Vimes is usually a person, not a team
    .saveScreenshot(directory + "3-Add Team screen.png")
    .click('button[class~=submitForm]')
    // Submit form
    .pause(1000)
    .assert.containsText('.Teams', 'Sam Vimes')
    // Check Teams includes Sam Vimes
    .saveScreenshot(directory + "4-Sam Vimes in the watch.png")

    .click('button[class~=addTeam]')
    .waitForElementVisible('input[id=TeamName]', 1000)
    .setValue('input[id=TeamName]', 'Findthee Swing')
    .click('button[class~=submitForm]')
    .pause(1000)
    // Make another team, also a person, who isn't in Night Watch, called Findthee
    Swing

    .assert.containsText('.Teams', 'Sam Vimes')
    .assert.containsText('.Teams', 'Findthee Swing')
    .saveScreenshot(directory + "5-2 teams.png")
    // Check Vimes is still there and that Swing is now
  },

'Deleting' : function (browser) {
  let directory = "./reports/screenshots/TestTeams/";
  browser
    .click('button[name=delete-findthee-swing]')
    // Delete Swing
    .pause(1000);
  browser.expect.element('.Teams').text.to.not.contain('Findthee Swing');
  browser.expect.element('.Teams').text.to.contain('Sam Vimes');
  // Check Vimes is there, Swing isn't
  browser
    .saveScreenshot(directory + "6-Deleted swing.png")
    .click('button[name=delete-sam-vimes]')
    .pause(1000);
  browser.expect.element('.Teams').text.to.not.contain('Sam Vimes');
  // Delete Vimes nad check he isn't there
  browser
    .url('http://localhost:8080/league')
    .click('button[name=delete-nights-watch]')
}

```

```

.pause(1000);
browser.expect.element('.Leagues').text.to.not.contain('Nights Watch');
// Delete League nad check it's really gone
browser.end(); // close browser
}
};

```

3.10.4. Tests at this point

3.11. 1. Teams Work

3.11.1. JUnit Tests

3.11.2. com.saskcow.bowling.repository.GameRepositoryTest

1 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.repository.GameRepositoryTest	thingsSaved_canBeRetrieved	0.067
com.saskcow.bowling.repository.GameRepositoryTest		0.067

3.11.3. com.saskcow.bowling.repository.LeagueRepositoryTest

3 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeRetrieved	0.011
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeQueried	0.146
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeDeleted	0.032
com.saskcow.bowling.repository.LeagueRepositoryTest		0.191

3.11.4. com.saskcow.bowling.BowlingApplicationTests

1 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.BowlingApplicationTests	contextLoads	0.046

Classname	Name	Time
com.saskcow.bowling.BowlingApplicationTests		0.046

3.11.5. com.saskcow.bowling.controller.TeamControllerTest

2 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.TeamControllerTest	deleteLeague_shouldDeleteLeague	0.036
com.saskcow.bowling.controller.TeamControllerTest	addLeague_shouldSaveTheCourse	0.076
com.saskcow.bowling.controller.TeamControllerTest		0.112

3.11.6. com.saskcow.bowling.controller.LeagueControllerTest

3 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.LeagueControllerTest	deleteLeague_shouldDeleteLeague	0.22
com.saskcow.bowling.controller.LeagueControllerTest	getLeague_shouldFilter	0.126
com.saskcow.bowling.controller.LeagueControllerTest	addLeague_shouldSaveTheCourse	0.115
com.saskcow.bowling.controller.LeagueControllerTest		0.462

3.11.7. Nightwatch Tests

3.11.8. TestTeams

Test Results

3.11.9. TestTeams

3 tests, 0 failed, 0 errors,

Classname	Name	Time	Assertions
TestTeams	Setup	4.978	3

Classname	Name	Time	Assertions
TestTeams	Test Adding Teams	4.600	6
TestTeams	Deleting	1.786	4
TestTeams		11.36	

Screenshots

5-2 teams

Nights Watch

Teams

- [Sam Vimes](#) 
- [Findthee Swing](#) 



6-Deleted swing

Nights Watch

Teams

- [Sam Vimes](#) 



3-Add Team screen

Add a Team to the League!

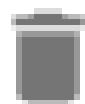
Team Name

Sam Vimes

SUBMIT

1-init league

- Nights Watch



REFRESH LEAGUES



2-League view

Nights Watch

Teams



Nights Watch

Teams

- [Sam Vimes](#)



+

3.11.10. TestLeague

Test Results

3.11.11. TestLeague

2 tests, 0 failed, 0 errors,

Classname	Name	Time	Assertions
TestLeague	Test adding leagues	7.651	5
TestLeague	Test removing leagues	1.648	3
TestLeague		9.299	

Screenshots

- nightwatch



REFRESH LEAGUES



4-Second League

- Nights Watch



- nightwatch



- daywatch



REFRESH LEAGUES



1-start

- Nights Watch



REFRESH LEAGUES

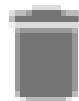


3-Shows League

- Nights Watch



- nightwatch



REFRESH LEAGUES



2-adding a league

Add a league!

League Name
nightwatch|

SUBMIT

3.11.12. Errors at this stage

CSS selectors are hard, so that took a while and sometimes patience is required to correctly assert.

Not waiting long enough

Sometimes the page hasn't loaded, so an element isn't seen, this can cause a click to fail, causing errors down the line as the test gets lost.

To fix this many `.pause(1000)` were put in after any redirects to ensure it doesn't end up missing something.

Incorrect CSS selector

CSS Selectors can be easy to have the wrong elements checked resulting in errors such as this.

```
[1 Creating Teams And Players] Test Suite
=====
Running: Add some Leagues
✓ Element <body> was visible after 71 milliseconds.
✗ Timed out while waiting for element <.Leagues> to be present for 2000 milliseconds. - expected "visible" but got: "not found"
  at Object.Add some Leagues (/home/saskcow/IdeaProjects/bowling/src/test/js/1. Creating Teams and Players.js:9:8)
  at process._tickCallback (internal/process/next_tick.js:176:11)

FAILED: 1 assertions failed and 1 passed (4.283s)
```

Figure 14. Expected visible, but not found

One example was `li[class=name] ⇒ li[class~=name] as it had other classes too, so this was necessary

Others include mistyping classes or ids in selectors.

3.12. Player Controller

Teams have players, as do games, so it seems a good place to go next. Again it'll start with tests then continue onwards.

3.12.1. PlayerControllerTest.java

This will be very similar to Team Controller as it does similar things.

```
package com.saskcow.bowling.controller;

import ...

@RunWith(MockitoJUnitRunner.class)
public class PlayerControllerTest {

    @Mock
    private PlayerRepository repo;
    @Mock
    private TeamRepository teamRepository;
    private MockMvc mockMvc;

    @Before
    public void setUp() {
        mockMvc = MockMvcBuilders.standaloneSetup(new PlayerController(repo,
teamRepository))
            .build();
    }

    @Test
    public void addPlayer_shouldSaveThePlayer() throws Exception {
        Team team = new Team(1L, "Brian's Bowlers!", new LinkedList<>(), null, null);
        Player player = new Player(1L, "Brian", team);
        // Create a team and a player, which isn't yet added to the team

        when(repo.save(isA(Player.class))).thenReturn(player);
        when(repo.findOne(player.getId())).thenReturn(player);
        when(repo.findAll()).thenReturn(Collections.singletonList(player));
        // Mock methods to return player when repo would return player

        when(teamRepository.findOne(team.getId())).thenReturn(team);
        // Mock to return team when team found in teamRepository

        String uri = mockMvc.perform(post("/api/player")
            // Send a post request to /api/player
            .content("{\"name\":\"Brian\", \"teamId\": \"1\"}")
            // with this data
            .contentType("application/json"))
            // and this content-type header
            .andExpect(status().isCreated())
            // Expect a 201
            .andExpect(header().string("Location",
"http://localhost:8080/api/player/" + player.getId()))
            // Expect Location to have path to object
    }
}
```

```

        .andReturn().getResponse().getHeader("Location");
        // Return object

    mockMvc.perform(get(uri))
        // Send a get request to the returned uri
        .andExpect(status().isOk())
        // Expect 200
        .andExpect(MockMvcResultMatchers.jsonPath("$.name", equalTo("Brian")))
        .andExpect(MockMvcResultMatchers.jsonPath("name", equalTo("Brian")));
        // Expect Player Name is Brian
    }

@Test
public void deletePlayer_shouldDeletePlayer() throws Exception {
    Team team = new Team(1L, "Brian's Bowlers!", new LinkedList<>(), null, null);
    Player brian = new Player(1L, "Brian", team);
    Player dave = new Player(2L, "Dave", team);
    team.addPlayer(brian);
    team.addPlayer(dave);
    // Create a team with 2 players

    doNothing().when(repo).delete(isA(Long.class));
    // when repo delete called, do nothing
    when(repo.findOne(brian.getId())).thenReturn(brian);
    // when Player brian found in repo, return him

    mockMvc.perform(delete("/api/player/1"))
        // Send the order to "delete" brian
        .andExpect(status().isNoContent());
        // Expect no content

    verify(repo, times(1)).delete(1L);
    // Check the act was done
    assertThat(team.getPlayers()).doesNotContain(brian);
    assertThat(team.getPlayers()).contains(dave);
    // Check dave didn't swap with brian, and is still there
}
}

```

3.12.2. PlayerController.java

Also similar to TeamController, but this one routes Player requests not Team requests.

```

package com.saskcow.bowling.controller;

import ...

@Controller // It's a controller, send requests here
public class PlayerController {

```

```

private PlayerRepository repo;
private TeamRepository teamRepository;
// Has a PlayerRepository and a TeamRepository

@Autowired
public PlayerController(PlayerRepository repo, TeamRepository teamRepository){
    this.repo = repo;
    this.teamRepository = teamRepository;
}
// Create one with a PlayerRepository and a TeamRepository from application
context

@RequestMapping(value = "/api/player/{id}", method = RequestMethod.GET)
// Get requests to /api/player/{id} call this function
public ResponseEntity<PlayerView> findLeague(@PathVariable("id") Long id) {
    Player player = repo.findOne(id);
    PlayerView playerView = new PlayerView(player);
    // Get a player, then make it a PlayerView
    return ResponseEntity.ok(playerView);
    // Return a 200 with the playerView
}

@RequestMapping(value = "/api/player", method = RequestMethod.POST)
// Post requests to /api/player/ call this function
public ResponseEntity<?> saveLeague(@RequestBody PlayerRest player) {
    // Get PlayerRest from JSON sent
    Team team = teamRepository.findOne(player.getTeamId());
    // Get the team from the teamId in playerRest
    Player savedPlayer = repo.save(new Player(player.getName(), team));
    // Save new Player with name and team
    team.addPlayer(savedPlayer);
    // Add to the team
    teamRepository.save(team);
    // Save the team
    URI location = ServletUriComponentsBuilder
        .fromCurrentRequest().path("/{id}")
        .buildAndExpand(savedPlayer.getId()).toUri();
    // Get the URI where one could retrieve this object
    return ResponseEntity.created(location).build();
    // Send the location as a header with a 200
}

@RequestMapping(value = "/api/player/{id}", method = RequestMethod.DELETE)
// Delete requests to /api/player/{id} call this function
public ResponseEntity<Void> deleteLeague(@PathVariable("id") Long id) {
    // Take the id from uri
    try {
        Player player = repo.findOne(id);
        player.getTeam().deletePlayer(player);
        // Remove the specified player from the team
        repo.delete(id);
    }
}

```

```

    // Delete the player
    return ResponseEntity.noContent().build();
    // Return 204
} catch (ResourceNotFoundException e) {
    return ResponseEntity.notFound().build();
    // If you can't find the player, send a 400
}
}
}

```

3.12.3. PlayerRest.java

We need to make players now, and the conventional methods don't send well in JSON. So this'll have to do.

```

package com.saskcow.bowling.rest;

import ...

@Data
@NoArgsConstructor
@AllArgsConstructor
public class PlayerRest {
    private String name;
    private Long teamId;
}

```

So it takes JSON

```
{
    name: "Player Name",
    teamId: 314
}
```

3.12.4. PlayerView.java

Now we need to view the player, as irritatingly enough sending back objects with these references still causes StackOverflow.

```

package com.saskcow.bowling.view;

import ...

@Data
@NoArgsConstructor
public class PlayerView {
    private Long id;
    private TeamViewSummary team;
    private String name;

    public PlayerView(Player player){
        this.id = player.getId();
        this.name = player.getName();
        this.team = new TeamViewSummary(player.getTeam());
    } // Player name and id, but team is a TeamViewSummary now
}

```

3.12.5. PlayerViewSummary.java

Once again, a name and an id

```

import ...

@Data
@NoArgsConstructor
public class PlayerViewSummary {
    private Long id;
    private String name;

    public PlayerViewSummary(Player player){
        this.id = player.getId();
        this.name = player.getName();
    }
}

```

3.12.6. Errors at this stage

Invalid Type

`Player player = new Team(...);` doesn't work as we cannot convert a Team to a Player. And it should be a Team anyway.

```

skcow/IdeaProjects/bowling/src/test/java/com/saskcow/bowling/controller/GameControllerTest.java:135: error: incompatible types: Team cannot be converted to Player
Player cableStreet = new Team(1L, "Cable Street Particulars", null, new LinkedList<>(), cityWatch, new LinkedList<>());

```

Figure 15. Incompatible Types

3.13. 2. Players back end only

3.13.1. JUnit Tests

3.13.2. com.saskcow.bowling.repository.GameRepositoryTest

1 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.repository.GameRepositoryTest	thingsSaved_canBeRetrieved	0.067
com.saskcow.bowling.repository.GameRepositoryTest		0.067

3.13.3. com.saskcow.bowling.repository.LeagueRepositoryTest

3 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeRetrieved	0.011
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeQueried	0.146
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeDeleted	0.032
com.saskcow.bowling.repository.LeagueRepositoryTest		0.191

3.13.4. com.saskcow.bowling.BowlingApplicationTests

1 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.BowlingApplicationTests	contextLoads	0.046
com.saskcow.bowling.BowlingApplicationTests		0.046

3.13.5. com.saskcow.bowling.controller.TeamControllerTest

2 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.TeamControllerTest	deleteLeague_shouldDeleteLeague	0.036

Classname	Name	Time
com.saskcow.bowling.controller.TeamControllerTest	addLeague_shouldSaveTheCourse	0.076
com.saskcow.bowling.controller.TeamControllerTest		0.112

3.13.6. com.saskcow.bowling.controller.LeagueControllerTest

3 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.LeagueControllerTest	deleteLeague_shouldDeleteLeague	0.22
com.saskcow.bowling.controller.LeagueControllerTest	getLeague_shouldFilter	0.126
com.saskcow.bowling.controller.LeagueControllerTest	addLeague_shouldSaveTheCourse	0.115
com.saskcow.bowling.controller.LeagueControllerTest		0.462

3.13.7. com.saskcow.bowling.controller.PlayerControllerTest

2 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.PlayerControllerTest	deleteLeague_shouldDeleteLeague	0.075
com.saskcow.bowling.controller.PlayerControllerTest	addLeague_shouldSaveTheCourse	0.036
com.saskcow.bowling.controller.PlayerControllerTest		0.112

3.14. Player Front End

3.14.1. App.js

Added

```
<Route exact path="/player/:id" component={Player}/>
<Route path={"/team/:id/add-player"} component={AddPlayer}/>
```

To give routes for adding and showing Player.

3.14.2. Team.js

Included in .Team to display all the players in the team.

```
<h3>Players</h3>
// Title Players
<ul className={"Players"}>
  // Create an unordered list class Players
  {this.state.players.map(player => (
    // For every player in this.state.players, return this
    <PlayerSummary key={player.id} id={player.id}>{player.name}</PlayerSummary>
    // Make a PlayerSummary for every player in this.state.players
  )))
</ul>
<Link to={"/team/" + this.state.id + '/add-player'}><Button className={"addPlayer"} variant={"fab"} color={"primary"}><AddIcon /></Button></Link>
// Link to the soon to be added add team page.
```

3.14.3. AddPlayer.js

We need to add players, so this will call the controller add player.

```
import ...

class AddPlayer extends React.Component {
  constructor(props) {
    super();
    this.state = ({name: "No Name", teamId: props.match.params.id});
    this.submit = this.submit.bind(this);
    this.updateName = this.updateName.bind(this);
    // Standard bind this to functions and set default state
  }

  submit(event) {
    event.preventDefault();
    // Prevent normal behaviour of a button
    axios.post("/api/player", {name: this.state.name, teamId: this.state.teamId})
    // Send a post request to /api/player with name and teamId, to match the
    PlayerRest object
    .then(response => {
      window.location.href = '/team/' + this.state.teamId;
      // Redirect to the team the player was made in
      this.state.name = "";
      console.log("created at " + response.headers.location);
    })
    .catch(function (error) {
      console.log(error);
      // Log any errors
    });
  }
}
```

```

}

updateName(event) {
  this.setState({name: event.target.value})
  // On event set this.state.name to value of what sent the event
}

render() {
  // When displayed in page
  return (
    <div className={"AddScreen"}>
      // Wrap it in a div .AddScreen
      <h1>Add a Team to the League!</h1>
      <form className={"thePlayerForm"} onSubmit={this.submit}>
        // Create a form which calls submit when submitted
        <TextField
          id="PlayerName"
          label="Player Name"
          placeholder="Player Name"
          className={"PlayerNameInput"}
          onChange={this.updateName}
        /> <br />
        // Text input which calls updateName when changed with label PlayerName
        <Button type={"submit"} variant={"raised"} color={"primary"}>
          className={"submitForm"}>Submit</Button>
          // Call submit
        </form>
      </div>
    )
  }
}

export default AddPlayer;

```

3.14.4. PlayerSummary.js

Team needs to show players, with PlayerSummarys, so we implement them.

```
import {Link} from "react-router-dom";

class PlayerSummary extends React.Component {
  render() {
    return (
      <li className={this.props.children.replace(/\s+/g, '-').toLowerCase()}>
        // Return a list element with class of player name
        <Link to={'/player/' + this.props.id.toString()}>
          {this.props.children}
        </Link>
        // Link to player page
        <Delete id={this.props.id} type={'player'} name={this.props.children}/>
        // Delete Button
      </li>
    )
  }
}

export default PlayerSummary;
```

3.14.5. Player.js

Display more details of the player.

```

import ...

class Player extends React.Component {

  constructor(props) {
    super();
    this.state = {status: "Loading"};
    this.getLeague = this.getLeague.bind(this);
    this.getLeague(props.match.params.id);
    // Set default state, bind this to getLeague and getLeague
  }

  getLeague(id) {
    axios.get('/api/player/' + id)
      // Send a get request to playerController and get the player details
      .then(response => {
        this.setState({
          status: "OK",
          id: id,
          name: response.data.name,
          team: response.data.team
          // Map response to state
        })
      })
      .catch(error => {
        console.log(error)
        // Simple error logging
      });
  }

  render() {
    if(this.state.status === "OK") {
      return (
        <div className='Player'>
          <h2>{this.state.name}</h2>
          <Link to={"/team/" + this.state.team.id}>{this.state.team.name}</Link>
        </div>
        // Show a title, and a link back
      )
    } else {
      return <CircularProgress color="primary" />
      // If loading show a loading wheel
    }
  }
}

export default Player;

```

3.14.6. TestPlayers.js

Now nightwatch is running, we need to add tests to test the new functionality.

```
module.exports = {
  // Export this

  'Setup' : function (browser) {
    // Create TestCase Setup
    let directory = "./reports/screenshots/TestPlayers/";
    browser
      .url('http://user:saskcow@localhost:8080/league')
      .pause(1000)
      .waitForElementVisible('body', 1000)
      .click('button[class~=add]')
      .waitForElementVisible('input[id=LeagueName]', 1000)
      .setValue('input[id=LeagueName]', 'Nights Watch')
      .click('button[class~=submitForm]')
      .pause(1000)
      .assert.containsText('.Leagues', 'Nights Watch')

      .click('li[class=nights-watch]>a')
      .pause(1000)
      .assert.containsText('h2', 'Nights Watch')

      .click('button[class~=addTeam]')
      .waitForElementVisible('input[id=TeamName]', 1000)
      .setValue('input[id=TeamName]', 'Sam Vimes')
      .click('button[class~=submitForm]')
      .pause(1000)
      .assert.containsText('.Teams', 'Sam Vimes')

      .saveScreenshot(directory + "1-init team.png")

    // Makes a League and Team like in the other examples
  },

  'Test Adding Players' : function (browser) {
    let directory = "./reports/screenshots/TestPlayers/";
    browser
      .click('li[class=sam-vimes]>a')
      .pause(1000)
      .assert.containsText('h2', 'Sam Vimes')
      .saveScreenshot(directory + "2-Team view.png")
    // Click on a team, and check it's the right one

      .click('button[class~=addPlayer]')
      .waitForElementVisible('input[id=PlayerName]', 1000)
    // Click add a player and wait for page to load
      .setValue('input[id=PlayerName]', 'Sam Vimes')
```

```

// Set player Name
.saveScreenshot(directory + "3-Add Player screen.png")
.click('button[class~=submitForm]')
.pause(1000)
// Submit player and wait for page to load
.assert.containsText('.Players', 'Sam Vimes')
.saveScreenshot(directory + "4-Sam Vimes in the Vimes.png")
// Check player exists

.click('button[class~=addPlayer]')
.waitForElementVisible('input[id=PlayerName]', 1000)
.setValue('input[id=PlayerName]', 'Mas Mives')
.click('button[class~=submitForm]')
.pause(1000)
.assert.containsText('.Players', 'Mas Mives')
.saveScreenshot(directory + "5-2 players.png")
// Add another player
},

'Test Player' : function (browser) {
let directory = "./reports/screenshots/TestPlayers/";
browser
.click('li[class=mas-mives]>a')
.pause(1000)
.assert.containsText('h2', 'Mas Mives')
.assert.containsText('a', 'Sam Vimes')
.saveScreenshot(directory + "6-Mas Mives.png")
.click('a')
// Look at a player, check a name and a back button appears, and leave
// Players don't have much going for them
},

'Deleting' : function (browser) {
let directory = "./reports/screenshots/TestPlayers/";
browser
.click('button[name=delete-mas-mives]')
.pause(1000);
browser.expect.element('.Players').text.to.not.contain('Mas Mives');
browser.expect.element('.Players').text.to.contain('Sam Vimes');
// Delete Mas Mives and check he's gone
browser
.saveScreenshot(directory + "7-Deleted Mives.png")
.click('button[name=delete-sam-vimes]')
.pause(1000);
browser.expect.element('.Players').text.to.not.contain('Sam Vimes');
// Delete Sam Vimes and check he's gone
browser
.url('http://localhost:8080/league')
.click('button[name=delete-nights-watch]')
.pause(1000);
browser.expect.element('.Leagues').text.to.not.contain('Nights Watch');
}

```

```

    // Delete the League
    browser.end();
    // Close the browser
}
};

```

3.15. 3.Players Front and Back

3.15.1. Nightwatch Tests

3.15.2. TestPlayers

Test Results

3.15.3. TestPlayers

4 tests, 0 failed, 0 errors,

Classname	Name	Time	Assertions
TestPlayers	Setup	7.439	6
TestPlayers	Test Adding Players	4.548	5
TestPlayers	Test Player	1.389	2
TestPlayers	Deleting	1.805	4
TestPlayers		15.18	

Screenshots

5-2 players

Sam Vimes

[Nights Watch](#)

Players

- [Sam Vimes](#) 
- [Mas Mives](#) 

+

3-Add Player screen

Add a Team to the League!

Player Name

Sam Vimes

SUBMIT

2-Team view

Sam Vimes

[Nights Watch](#)

Players

+

1-init team

Nights Watch

[Back](#)

Teams

- [Sam Vimes](#) 



4-Sam Vimes in the Vimes

Sam Vimes

[Nights Watch](#)

Players

- [Sam Vimes](#) 



6-Mas Mives

Mas Mives

[Sam Vimes](#)

7-Deleted Mives

Sam Vimes

[Nights Watch](#)

Players

- [Sam Vimes](#)

+

3.15.4. TestTeams

Test Results

3.15.5. TestTeams

3 tests, 0 failed, 0 errors,

Classname	Name	Time	Assertions
TestTeams	Setup	4.924	3
TestTeams	Test Adding Teams	4.571	6
TestTeams	Deleting	1.790	4
TestTeams		11.29	

Screenshots

5-2 teams

Nights Watch

[Back](#)

Teams

- [Sam Vimes](#) 
- [Findthee Swing](#) 



6-Deleted swing

Nights Watch

[Back](#)

Teams

- [Sam Vimes](#) 



3-Add Team screen

Add a Team to the League!

Team Name

Sam Vimes

SUBMIT

1-init league

- Nights Watch



REFRESH LEAGUES



2-League view

Nights Watch

[Back](#)

Teams



4-Sam Vimes in the watch

Nights Watch

[Back](#)

Teams

- [Sam Vimes](#)



3.15.6. TestLeague

Test Results

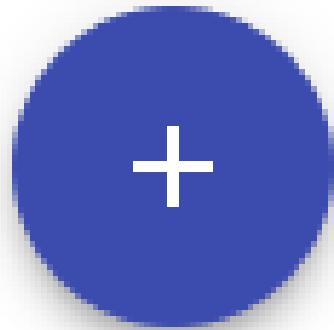
3.15.7. TestLeague

2 tests, 0 failed, 0 errors,

Classname	Name	Time	Assertions
TestLeague	Test adding leagues	7.707	5
TestLeague	Test removing leagues	1.584	3
TestLeague		9.291	

Screenshots

5-Deleted daywatch



4-Second League

- nightwatch



- daywatch



REFRESH LEAGUES



1-start

REFRESH LEAGUES



3-Shows League

- nightwatch



REFRESH LEAGUES



2-adding a league

Add a league!

League Name
nightwatch

SUBMIT

3.16. Games in Back end

Game is a bit different, teams are a oneToMany with League, players are a manyToMany with Team, games have oneToMany with Team and League. Despite this this is still fairly similar.

3.16.1. GameControllerTest.java

Got to test the controller to save time later on.

```
package com.saskcow.bowling.controller;  
  
import ...  
  
@RunWith(MockitoJUnitRunner.class)
```

```

public class GameControllerTest {

    @Mock
    private GameRepository repo;
    @Mock
    private TeamRepository teamRepository;
    // Create a mock GameRepository, TeamRepository
    private MockMvc mockMvc;

    @Before
    public void setUp() {
        mockMvc = MockMvcBuilders.standaloneSetup(new GameController(repo,
teamRepository))
                .build();
    }
    // Create a mockMvc Controller from a new GameController with the mock repos

    @Test
    // It's a test, run it with gradle test, and report on it
    public void addGame_shouldSaveTheGame() throws Exception {
        League league = new League(1L, "Brian", null, new LinkedList<>());
        Team team1 = new Team(1L, "Dave", null, new LinkedList<>(), league);
        Team team2 = new Team(2L, "David", null, new LinkedList<>(), league);
        LocalDateTime dateTime = LocalDateTime.now(Clock.systemUTC());
        // Create LocalDateTime from current time
        Game game = new Game(1L, dateTime, "Brian Bowling Centre", new
LinkedList<>(Arrays.asList(team1, team2)));

        // Create a bunch of objects which would be there when saving a game

        team1.addGame(game);
        team2.addGame(game);
        league.addTeam(team1);
        league.addTeam(team2);

        // Link some of these together

        when(repo.save(isA(Game.class))).thenReturn(game);
        when(repo.findOne(game.getId())).thenReturn(game);
        when(repo.findAll()).thenReturn(Collections.singletonList(game));
        when(teamRepository.findOne(team1.getId())).thenReturn(team1);
        when(teamRepository.findOne(team2.getId())).thenReturn(team2);

        // Mock what the repository would do, return objects when find attempted

        DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME;
        // Make a formatter which takes a LocalDateTime and makes it an ISO local date
        time string
        String content = String.format("{\"time\":\"%s\", \"venue\": \"Brian Bowling
Centre\", \"teamId1\": \"1\", \"teamId2\": \"2\"}", dateTime.format(formatter));
        // Create the content for the GameRest , time, venue, teamId1, teamId2
    }
}

```

```

// String.format() swaps the %s for the formatted LocalDateTime String

String uri = mockMvc.perform(post("/api/game")
        // Send a post request to /api/game
        .content(content)
        // Send the JSON String above
        .contentType("application/json"))
        // Include content-type header
        .andExpect(status().isCreated())
        // Expect 201 returned
        .andExpect(header().string("Location",
"http://localhost:8080/api/game/" + game.getId()))
        // Expect header of Location of game
        .andReturn().getResponse().getHeader("Location");
        // Return Location

mockMvc.perform(get(uri))
        .andExpect(status().isOk())
        // Expect 200
        .andExpect(MockMvcResultMatchers.jsonPath("$.venue", equalTo("Brian
Bowling Centre")))
        .andExpect(MockMvcResultMatchers.jsonPath("venue", equalTo("Brian
Bowling Centre")))
        // Expect Venue same as set in game, Brian Bowling Centre
        .andExpect(MockMvcResultMatchers.jsonPath("$.league.name",
equalTo("Brian")))
        .andExpect(MockMvcResultMatchers.jsonPath("$.time",
equalTo(dateTime.format(formatter)))); 
        // Expect League Name and time to be as in game
}

@Test
public void deleteGame_shouldDeleteGame() throws Exception {
    League league = new League(1L, "Brian", null, new LinkedList<>());
    Team team1 = new Team(1L, "Brian", null, new LinkedList<>(), league);
    Team team2 = new Team(2L, "Brian", null, new LinkedList<>(), league);
    LocalDateTime dateTime = LocalDateTime.now(Clock.systemUTC());
    // Create LocalDateTime from current time
    Game game = new Game(1L, dateTime, "Brian Bowling Centre", new
LinkedList<>(Arrays.asList(team1, team2)));
    Game game2 = new Game(2L, dateTime, "Brian Bowling Centre", new
LinkedList<>(Arrays.asList(team1, team2)));

    // Create objects which would be in place

    team1.addGame(game);
    team2.addGame(game);
    team1.addGame(game2);
    team2.addGame(game2);
    league.addTeam(team1);
    league.addTeam(team2);
}

```

```

    // Link all the objects

    doNothing().when(repo).delete(isA(Long.class));
    when(repo.findOne(game.getId())).thenReturn(game);

    // When delete attempted do nothing, when findOne of gameId return game

    mockMvc.perform(delete("/api/game/1"))
        // Send Delete request to /api/game/1
        .andExpect(status().isNoContent());
        // Expect 204

    verify(repo, times(1)).delete(1L);
    // Check repo method called once
    assertThat(team1.getGames()).doesNotContain(game);
    assertThat(team1.getGames()).contains(game2);
    // Check that there are no links to game, but still to game2
}

}

```

3.16.2. GameController.java

What requests to /api/game go through, makes, deletes and modifies objects in database.

```

package com.saskcow.bowling.controller;

import ...

@Controller
public class GameController {
    private GameRepository repo;
    private TeamRepository teamRepository;

    @Autowired
    public GameController(GameRepository repo, TeamRepository teamRepository){
        this.repo = repo;
        this.teamRepository = teamRepository;
    }

    // The Usual has Repositories, and on start gets them from application context

    @RequestMapping(value = "/api/game/{id}", method = RequestMethod.GET)
    // All get requests to /api/game/{id} calls this function
    public ResponseEntity<GameView> findLeague(@PathVariable("id") Long id) {
        // Returns a ResponseEntity with a GameView, takes id from path
        Game game = repo.findOne(id);
        GameView gameView = new GameView(game);
        // Get the game and
    }
}

```

```

    return ResponseEntity.ok(gameView);
}

@RequestMapping(value = "/api/game", method = RequestMethod.POST)
// All post requests to /api/game/ calls this function
public ResponseEntity<?> saveLeague(@RequestBody GameRest game) {
    // Returns a ResponseEntity takes JSON as GameRest

    Team team1 = teamRepository.findOne(game.getTeamId1());
    Team team2 = teamRepository.findOne(game.getTeamId2());
    List<Team> teams = new LinkedList<>(Arrays.asList(team1, team2));
    DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME;
    LocalDateTime dateTime = LocalDateTime.parse(game.getTime(), formatter);
    // Find all the objects

    Game savedGame = repo.save(new Game(dateTime, game.getVenue(), teams));
    // Save the game
    team1.addGame(savedGame);
    team2.addGame(savedGame);
    // Tell everything else the Game exists
    teamRepository.save(team1);
    teamRepository.save(team2);
    // Save everything

    URI location = ServletUriComponentsBuilder
        .fromCurrentRequest().path("/{id}")
        .buildAndExpand(savedGame.getId()).toUri();
    // Put location in the URI
    return ResponseEntity.created(location).build();
    // Send a 201 with the location
}

@RequestMapping(value = "/api/game/{id}", method = RequestMethod.DELETE)
// All delete requests to /api/game/{id} calls this function
public ResponseEntity<Void> deleteLeague(@PathVariable("id") Long id) {
    // Return a Response, with nothing
    try {
        Game game = repo.findOne(id);
        game.getTeams().forEach(team -> team.deleteGame(game));
        // For each Team in the Game, remove the link
        repo.delete(id);
        // Delete the Game
        return ResponseEntity.noContent().build();
        // Send a 204
    } catch (ResourceNotFoundException e) {
        return ResponseEntity.notFound().build();
        // If you can't find the game, send back a 404
    }
}
}

```

3.16.3. GameRest.java

Once again, no foreign keys etc from JSON so we compromise.

```
package com.saskcow.bowling.rest;

import ...

@Data
@NoArgsConstructor
@AllArgsConstructor
public class GameRest {
    private String time;
    private String venue;
    private Long rotaId;
    private Long teamId1;
    private Long teamId2;
    // All we need to construct a Game
}
```

JSON Equivalent

```
{
    time: "2018-04-16T13:48:16",
    venue: "The Shades",
    rotaId: 3,
    teamId1: 1,
    teamId2: 4
}
```

3.16.4. GameView.java

This is the only view object for a game, as at this stage, there isn't much in a game you don't care about, which teams are playing, when and where are all fairly crucial. But we need at least one view object, so this is it!

```

package com.saskcow.bowling.view;

import ...

@Data
@NoArgsConstructor
public class GameView {
    private Long id;
    private String time;
    private List<TeamViewSummary> teams;
    private String venue;
    private LeagueViewSummary league;
    // Has an id, a time, 2 teams, a venue, and details of the league

    public GameView(Game game){
        this.id = game.getId();
        this.time = game.getTime().format(DateTimeFormatter.ISO_LOCAL_DATE_TIME);
        // Get time and format it to ISO local time string
        this.teams = game.getTeams().stream().map(TeamViewSummary::new).
        collect(Collectors.toList());
        // Make all the teams into TeamViewSummary and collect stream into a list
        this.venue = game.getVenue();
        this.league = new LeagueViewSummary(game.getTeams().get(0).getLeague());
        // Get League from team 1's League
        // Get the things from game
    }
}

```

3.16.5. Errors at this stage

@NoArgsConstructor Required

Originally the `@NoArgsConstructor` was not on GameRest. And that may sound fine, why would it be empty? Well as it turns out when converting the JSON, it needs an empty GameRest. That's easy enough to add.

```
Request processing failed; nested exception is org.springframework.http.converter.HttpMessageConversionException: Type definition error:
at [Source: (PushbackInputStream); line: 1, column: 2]
```

Figure 16. No noArgsConstructor

Now with the `@NoArgsConstructor`, which creates an empty constructor, it works.

3.17. 4. Games in back end

3.17.1. JUnit Tests

3.17.2. com.saskcow.bowling.repository.GameRepositoryTest

1 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.repository.GameRepositoryTest	thingsSaved_canBeRetrieved	0.087
com.saskcow.bowling.repository.GameRepositoryTest		0.087

3.17.3. com.saskcow.bowling.repository.LeagueRepositoryTest

3 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeRetrieved	0.013
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeQueried	0.204
com.saskcow.bowling.repository.LeagueRepositoryTest	thingsSaved_canBeDeleted	0.039
com.saskcow.bowling.repository.LeagueRepositoryTest		0.259

3.17.4. com.saskcow.bowling.BowlingApplicationTests

1 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.BowlingApplicationTests	contextLoads	0.049
com.saskcow.bowling.BowlingApplicationTests		0.049

3.17.5. com.saskcow.bowling.controller.TeamControllerTest

2 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.TeamControllerTest	addTeam_shouldSaveTheTeam	0.05
com.saskcow.bowling.controller.TeamControllerTest	deleteTeam_shouldDeleteTeam	0.029
com.saskcow.bowling.controller.TeamControllerTest		0.08

3.17.6. com.saskcow.bowling.controller.LeagueControllerTest

3 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.LeagueControllerTest	addLeague_shouldSaveTheLeague	0.097
com.saskcow.bowling.controller.LeagueControllerTest	deleteLeague_shouldDeleteLeague	0.03
com.saskcow.bowling.controller.LeagueControllerTest	getLeague_shouldFilter	0.049
com.saskcow.bowling.controller.LeagueControllerTest		0.177

3.17.7. com.saskcow.bowling.controller.PlayerControllerTest

2 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.PlayerControllerTest	deletePlayer_shouldDeletePlayer	0.031
com.saskcow.bowling.controller.PlayerControllerTest	addPlayer_shouldSaveThePlayer	0.057
com.saskcow.bowling.controller.PlayerControllerTest		0.089

3.17.8. com.saskcow.bowling.controller.GameControllerTest

2 tests, 0 failed, 0 errors,

Classname	Name	Time
com.saskcow.bowling.controller.GameControllerTest	addGame_shouldSaveTheGame	0.322
com.saskcow.bowling.controller.GameControllerTest	deleteGame_shouldDeleteGame	0.165
com.saskcow.bowling.controller.GameControllerTest		0.488

3.18. Games in the front end

3.18.1. App.js

Now we have the standard rewrite the router, we need to add the game and addGame pages.

```

<Route exact path="/" component={LeagueList}/>
<Route exact path="/league" component={LeagueList}/>
<Route exact path="/league/:id" component={League}/>
<Route exact path="/team/:id" component={Team}/>
<Route exact path="/player/:id" component={Player}/>
<Route path={"/league/:id/add-team"} component={AddTeam}/>
<Route path={"/league/:id/add-game"} component={AddGame}/>
<Route path={"/team/:id/add-player"} component={AddPlayer}/>
<Route path="/add/league" component={AddLeague}/>

```

Routes are now this, this will be like this for a while.

3.18.2. League.js

Games need to show somewhere, so a League can show all the games.

```

<h3>Games</h3>
<ul className={"Games"}>
  {this.state.games.map(game => (
    <GameSummary key={game.id} id={game.id} time={game.time}
    teams={game.teams}>{game.venue}</GameSummary>
  ))}
</ul>
<Link to={"/league/" + this.state.id + '/add-game'}><Button className={"addGame"}
  variant={"raised"} color={"primary"}>Add A Game</Button></Link>

```

3.18.3. AddGame.js

Now with more menus than ever (more than just a name) we have the next generation of add screen, AddGame.js!

```

import ...

class AddGame extends React.Component {
  constructor(props) {
    super();
    this.state = ({status: "Loading", time: new Date().toISOString().slice(0, -5)});
    // set default state, slice end of time string to remove the time zone part
    this.submit = this.submit.bind(this);
    this.updateVenue = this.updateVenue.bind(this);
    this.updateTime = this.updateTime.bind(this);
    this.updateTeam1 = this.updateTeam1.bind(this);
    this.updateTeam2 = this.updateTeam2.bind(this);
    this.getLeague = this.getLeague.bind(this);
    this.getLeague(props.match.params.id);
    // bind this to most things, and then getLeague
  }
}

```

```

submit(event) {
  event.preventDefault();
  // Prevent standard event behaviour
  axios.post("/api/game", {
    venue: this.state.venue,
    time: this.state.time,
    teamId1: this.state.team1,
    teamId2: this.state.team2,
    rotaId: this.state.rota
    // Send all this off as a GameRest style bit of JSON
  })
  .then(response => {
    window.location.href = '/league/' + this.state.id;
    this.state.name = "";
    console.log("created at " + response.headers.location);
    // Go to league page if done
  })
  .catch(function (error) {
    console.log(error);
    // log any errors into console
  });
}

getLeague(id) {
  axios.get('/api/league/' + id)
  // Send get request to /api/league/{id} to get league data, says which teams there
  are etc.
  .then(response => {
    this.setState({
      id: id,
      status: "OK",
      teams: response.data.teams,
      leagueName: response.data.name,
      team1: response.data.teams[0].id,
      team2: response.data.teams[1].id,
      rota: response.data.rotaId
      // Map response to state
    })
  })
  .catch(error => {
    console.log(error)
    // Log any errors in console
  });
}

updateVenue(event) {
  this.setState({venue: event.target.value});
}

updateTime(event) {

```

```

        console.log(event.target.value);
        this.setState({time: event.target.value});
    }

updateTeam1(event) {
    this.setState({team1: event.target.value});
};

updateTeam2(event) {
    this.setState({team2: event.target.value});
};

// All these just update state as sent an event, like all the others, but there are
4 this time

render() {
    if(this.state.status === "OK") {
        // If loaded
        return (
            <div className={"AddScreen"}>
                <h1>Add game {this.state.name} to {this.state.leagueName}!</h1>
                <form className={"theGameForm"} onSubmit={this.submit} noValidate>
                    // Create a form which calls submit on finish, do not try to validate the
dropdowns

                    <TextField
                        id="Venue"
                        label="Venue"
                        placeholder="Venue"
                        className={"VenueInput"}
                        onChange={this.updateVenue}
                    />
                    // Venue text entry

                    <br />
                    <br />
                    // Spacing, so it's not one wide webpage, but has a column

                    <TextField
                        id="datetime-local"
                        type="datetime-local"
                        className={"TimeInput"}
                        defaultValue={this.state.time}
                        onChange={this.updateTime}
                    />
                    // Time picker, which is technically a string, which is rather convenient

                    <br />
                    <br />

                    <InputLabel htmlFor="team-1">team1</InputLabel>

```

```

<Select
  value={this.state.team1}
  // Start value is this.state.team1
  onChange={this.updateTeam1}
  // When changed call updateTeam1 with event
  id='team1'
  className={'team1'}
  inputProps={{
    name: 'team1',
    id: 'team1',
  }}>
  {this.state.teams.map(team => (
    <MenuItem key={team.id} value={team.id} name={"team1-" +
team.name.replace(/\s+/g, '-').toLowerCase()}>{team.name}</MenuItem>
  ))}
  // For each team, make a menu item with a unique value and a name
</Select>
// Select from dropdown of MenuItems onChange set team1 in state to the
correct Team

```

```

<br />
<br />

<InputLabel htmlFor="team-2">team2</InputLabel>
<Select
  value={this.state.team2}
  // Start value is this.state.team2
  onChange={this.updateTeam2}
  // When changed call updateTeam2 with event
  id='team2'
  className={'team2'}
  inputProps={{
    name: 'team2',
    id: 'team2',
  }}>
  {this.state.teams.map(team => (
    <MenuItem key={team.id} value={team.id} name={"team2-" +
team.name.replace(/\s+/g, '-').toLowerCase()}>{team.name}</MenuItem>
  ))}
  // For each team, make a menu item with a unique value and a name
</Select>
// Select from dropdown of MenuItems onChange set team1 in state to the
correct Team

```

```

<br />
<br />

<Button type="submit" variant="raised" color="primary"
className="submitForm">Submit</Button>
// Submit form

```

```

        </form>
    </div>
)
} else {
// If not loaded, send progress wheel
return (
<CircularProgress />
)
}
}

export default AddGame;

```

Well that was certainly longer than the rest, the Rest objects were short.

3.18.4. GameSummary.js

If the League is going to display Games, we need the summary, as games don't have a name or something, we require a lot more detail.

```

import ...

class GameSummary extends React.Component {
render() {
return (
<li>
<Link to={'/team/' + this.props.teams[0].id}>{this.props.teams[0].name}</Link>
vs
// Link to the first team in the game, with the team name
<Link to={'/team/' + this.props.teams[1].id}>{this.props.teams[1].name}</Link>
// Link to the second team in the game, also with name
<p>{this.props.children} - {this.props.time}</p>
// Linebreak, then the venue - time of the game
</li>
)
}
}

export default GameSummary; // Export this for easier imports

```

3.18.5. Errors at this stage

Updating wrong value in state

With addGame and it's 4 update state functions, there was a problem with the team updaters.

```
updateTeam1(event) {
    this.setState({team1: event.target.value});
};

updateTeam2(event) {
    this.setState({team1: event.target.value});
};
```

Both were updating team 1, causing the lower dropdown to change the upper dropdown. This wasn't ideal. So when spotted it was simply updated to team 1 and team 2 respectively, to fix this problem.

include::test/5. Games Front and Back.adoc

4. Stage 2

4.1. Planning

5. Test Formatter

I wrote a test formatter in python to format the xml into asciidoc for the writeup, can't be doing all this manually now can I?

5.1. Research

5.2. Planning

5.3. Development

5.4. Testing

path relative to file which imported it, not file which link originally in.

6. Testing

It is important to do testing throughout the project, screenshots and logs start from an early stage in development, test data logs begin when basic functionality starts. Most of this is with the associated version.

7. Api Docs

Api details on how mappings work and descriptions.

7.1. League API

7.1.1. Create A League

```
POST /api/league HTTP/1.1
```

Content-Type: application/json

Host: localhost:8080

Content-Length: 16

```
{"name": "Brian"}
```

HTTP/1.1 201 Created

Location: http://localhost:8080/api/league/1

Object sent usually will just be a name to create a league off, and all other objects will be created off of league.

7.1.2. Get a list of Leagues

```
GET /api/league HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Content-Length: 25

```
[{"id": 1, "name": "Brian"}]
```

Get's a summary of ALL leagues, returns as a list of "LeagueViewSummary" objects

7.1.3. Get one League

```
GET /api/league/1 HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Content-Length: 206

```
{"id": 1, "name": "Brian", "rotaId": null, "teams": [{"id": null, "name": "Team Brian", "pinsFor": 0, "pinsAgainst": 0, "highHandicapGame": 0, "highHandicapSeries": 0, "teamPoints": 0, "totalPoints": 0, "numGames": 0}], "games": []}
```

Gets one league based off of it's id. Single League gives more detail, returns a "LeagueView" not a "LeagueViewSummary"

7.1.4. Get a list of leagues by name

```
GET /api/league?name=Bri HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Content-Length: 25

```
[{"id":3,"name":"Brian"}]
```

The same as getting a list of all leagues, but will only show leagues with matching criteria. Matches off contains not exact match.

7.1.5. Delete A league

```
DELETE /api/league/1 HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 204 No Content

Also removes any teams associated with the league.

7.2. Team API

The Team API is simpler than the league API due to a "LeagueView" returning a list of "TeamViewSummary" already, this makes the use of a list redundant for the application.

7.2.1. Create A Team

```
POST /api/team HTTP/1.1
```

Content-Type: application/json

Host: localhost:8080

Content-Length: 33

```
{"name":"Brian", "leagueId": "1"}
```

HTTP/1.1 201 Created

Location: http://localhost:8080/api/team/1

The team creation takes a name for the team and a leagueid, to create a team with the league.

7.2.2. Get A Team

```
GET /api/team/1 HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Content-Length: 187

```
{"league": {"id": 1, "name": "Brian"}, "id": 1, "name": "Brian", "players": null, "games": null, "pinsFor": 0, "pinsAgainst": 0, "highHandicapGame": 0, "highHandicapSeries": 0, "teamPoints": 0, "totalPoints": 0}
```

This returns one team based off of ID, a list is not necessary due to a league giving a list.

Returns a "TeamView"

7.2.3. Delete A Team

```
DELETE /api/team/1 HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 204 No Content

Deletes the team, does not delete if the team has games.

7.3. Player API

Similar to team API as a team gives a list of players

7.3.1. Create A Player

```
POST /api/player HTTP/1.1
```

Content-Type: application/json

Host: localhost:8080

Content-Length: 31

```
{"name": "Brian", "teamId": "1"}
```

HTTP/1.1 201 Created

Location: http://localhost:8080/api/player/1

The player creation takes a name for the player and a teamid, to create a player with the team.

7.3.2. Get A Player

```
GET /api/player/1 HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Content-Length: 151

```
{"id":1,"team":{"id":1,"name":"Brian's Bowlers!"}, "name": "Brian", "handicap": 0  
, "recentGames": null, "highGame": 0, "lowGame": 0, "highSeries": 0, "lowSeries": 0}
```

This returns one player based off of ID, a list is not necessary due to a team giving a list.
Returns a "PlayerView"

7.3.3. Delete A Player

```
DELETE /api/player/1 HTTP/1.1
```

Host: localhost:8080

HTTP/1.1 204 No Content

Deletes the player.

7.4. Game API

Games between 2 teams, at a given venue and time

7.4.1. Create A Game

```
POST /api/game HTTP/1.1
```

Content-Type: application/json

Host: localhost:8080

Content-Length: 112

```
{"time": "2018-04-14T19:51:13.2", "venue": "Brian Bowling Centre", "rotaId": "1",  
"teamId1": "1", "teamId2": "2"}
```

HTTP/1.1 201 Created

Location: http://localhost:8080/api/game/1

The game creation takes a time (ISO local datetime string) and a venue (string) as well as ids for both teams involved in the game, upcoming games are not currently supported

7.4.2. Get A Game

```
GET /api/game/1 HTTP/1.1
```

```
Host: localhost:8080
```

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Content-Length: 177

```
{"id":1,"time":"2018-04-14T19:51:13.2","teams":[{"id":1,"name":"Dave"}, {"id":2,"name":"David"}],"venue":"Brian Bowling Centre","league":{"id":1,"name":"Brian"}, "playerGames":[]}
```

This returns one game based off of ID, a list is not necessary due to a team giving a list.

Returns a "GameView" should rarely be used as a league and a team shows games as "GameView"

7.4.3. Delete A Game

```
DELETE /api/game/1 HTTP/1.1
```

```
Host: localhost:8080
```

HTTP/1.1 204 No Content

Deletes the game.

7.4.4. Complete A Game

```
POST /api/game/1 HTTP/1.1
```

```
Host: localhost:8080
```

HTTP/1.1 204 No Content

Scores the game so one team can win, adds points which count towards team and player stats.

7.5. PlayerGame

A playergame is what is created when a player is added to play in a game.

7.5.1. Create 3 PlayerGames

```
POST /api/playergame HTTP/1.1
Content-Type: application/json
Host: localhost:8080
Content-Length: 54

{"playerIds": [1, 2, 3], "teamId": "1", "gameId": "1"}
```

```
HTTP/1.1 201 Created
Location: http://localhost:8080/api/playergame/1
```

Creates 3 PlayerGames.

7.5.2. Delete A PlayerGame

```
DELETE /api/playergame/1 HTTP/1.1
Host: localhost:8080
```

```
HTTP/1.1 204 No Content
```

Deletes one PlayerGame.

7.6. Score API

A Score is what contains a players scratch, handicap and points for each set in a game.

7.6.1. Create A Score

```
POST /api/playergame HTTP/1.1
Content-Type: application/json
Host: localhost:8080
Content-Length: 54

{"playerIds": [1, 2, 3], "teamId": "1", "gameId": "1"}
```

```
HTTP/1.1 201 Created
Location: http://localhost:8080/api/playergame/1
```

Creates a Score for a playerGame.

7.6.2. Delete A Score

```
DELETE /api/playergame/1 HTTP/1.1
```

```
Host: localhost:8080
```

```
HTTP/1.1 204 No Content
```

Deletes the Score.