Basketify Testing Plan

Zachary Schmidt, Aron Gu, Muhammed Rayyan Rashid

Testing Plan

Basketify Description

Basketify is a full-stack web application which allows users to view past NBA statistics for players and teams, as well as see our ML predictions for upcoming games. The primary tech stack is React for the frontend, Django for the backend, and MongoDB for the database. In order to test Basketify, we've broken down the main functionalities specified in the SRS into sections:

- **Dashboard**: tests involve making sure all components render correctly, pre-fetching connection to the database works as intended, and clicking on each button navigates to the correct page. Mostly frontend tests.
- **Search interface**: backend tests ensure names passed as player/team name produce an appropriate, well-formatted response from the backend. Frontend tests ensure matching names are actually displayed and clickable, and that the switch from player to team search works as intended (request sent to appropriate backend route).
- **Favourite player & team**: frontend tests involve ensuring the favorited player/team are displayed on the dashboard if available (blank if not), and that the search interface page sends a request to api/set-favorite/ when a name is clicked. Backend tests ensure get-favorite and set-favorite routes correctly interact with the PostgreSQL database and return valid HTTP responses.
- **User registration & login**: Backend tests ensure the registration and authentication endpoints work correctly, including email verification, password reset, and email change functionality. These tests verify the Django models and serializers are working as expected, testing both valid and invalid user inputs. JWT token generation and validation are also tested, along with the custom email verification system with its 2-minute expiration window. Frontend tests verify the registration and login forms display correctly, validation messages appear for incorrect inputs, and that successful authentication redirects users appropriately. These tests also check that password reset and email change workflows function properly, with appropriate success and error messages displayed to the user.
- **Stats table view**: frontend tests ensure 2 tables are displayed (with valid values: not empty or NaN or negative) when the page is loaded, buttons to switch from game-by-game and seasonal views render the corresponding tables. Backend tests ensure the player/stats/ and team/stats/ routes return JSON objects which have valid data for the player/team.
- **Stats graph view**: frontend tests ensure clicking on the buttons adds a new series to the graph, the graph is visible when >0 stats are selected, only 2 stats can be selected at a

time, de-selecting a button removes the corresponding series, and seasonal button switches the mode while stats selections stay. No backend tests: already covered by the stats table section.

- **Filtering of stats table**: Backend tests focus on verifying that the filtering utility functions correctly transform game data according to specified filter parameters. Tests ensure filters for date ranges, opponents, season types, conferences, divisions, and game outcomes all modify query results as expected. These tests also check that combinations of multiple filters work correctly together. Frontend tests verify that the filter UI renders correctly, opens and closes as expected, and that applying filters updates the displayed data without requiring page reload. Tests ensure filter status indicators are displayed when active, and that clearing filters resets the view appropriately. These tests also check that filters behave consistently across both player and team statistics views.
- **ML predictions:** Backend tests verify the format of predictions returned to the frontend are well-formed and can handle errors gracefully. Frontend tests verify the stats are displayed in a separate table on the StatsPage.js page.

Testing Approach

We sought to achieve statement coverage at a minimum, and ideally branch coverage of all user-written Python and JS code (this doesn't include code that comes built-in to a Django server such as the manage.py file). We are easily able to check our success in doing so using the testing tools described in the methodology section below.

We primarily focused on testing individual functions via unit testing, with some integration tests included and labelled in our test suite as well. Since our app is functional (not object-oriented) and the Django server is RESTful (i.e. stateless), it was relatively easy to come up with inputs to test each function with.

For acceptance-level testing of the FRs promised in our SRS, we had two approaches. Wherever possible, we tried to use automated frontend testing using the Jest library. Jest allowed us to render a page, click on buttons/input user data, and check what changed on the page. Mocking was kept to a minimum, primarily mocking the axios/api objects so that any requests to the Django backend would return mock data. These tests are listed in the Automated Frontend Tests table (and run with "npm test"). However, for some features like graphs, doing automated testing required prohibitive amounts of effort we thought was better spent on the project. For these features, we created a test case table a user could manually perform and check against the expected outcome in the table to verify the test. These are listed in the Manual Acceptance Tests table.

Generally, our unit and integration tests are forms of white box testing, while our acceptance-level tests are types of black box testing. Using both strategies allowed us to test both the inner workings of our code as well as the overall product in comparison to the SRS.

Testing Methodology

Fortunately, both React and Django have easy-to-use, built-in support for testing. Django allows developers to write tests using the *unittest* Python library, and to run them using the manage.py file. React uses a test runner called *Jest*, and tests can be executed using NodeJS.
Manual acceptance-level testing was accomplished using a test case table, which a tester is meant to manually follow and visually check if the result is the same as specified in the table.

Test Suite Execution Instructions

To execute the backend Django tests, from the root of the project repository:

(Follow all instructions in RUNNING_INSTRUCTIONS.txt, also attached below for convenience). Then:

```
$ cd test_server/backend
$ python3 manage.py test
```

To execute the frontend Jest tests, from the root of the project repository:

```
$ cd test_server/frontend
$ npm install --save-dev @testing-library/react @babel/preset-env @babel/preset-react babel-jest
$ npm test
```

(To see detailed coverage statistics, run it as: npm test -- --coverage

To execute ML model tests, from the root of the project repository:

```
$ cd test_server/pull_data_scripts/ml
$ python3 test_player_pred.py
$ python3 test_feedback_loop.py
```

To execute the acceptance-level tests, run the project (see test_server/RUNNING_INSTRUCTIONS.txt for details), and perform the actions as specified in the table below, then compare the expected output to your actual output.

Copy of RUNNING_INSTRUCTIONS.txt:
These instructions were tested and verified to work on a fresh Ubuntu 20.04 VM.

- Install Python 3.13:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get install python3.13-full
$ sudo apt-get install python3.13-dev
```

Install pip:

```
$ python3.13 -m ensurepip
```

Create & activate virtual environment:

```
$ python3.13 -m venv .venv
$ source .venv/bin/activate
```

- Install nvm (package manager that makes nodejs install easy):

```
$ sudo apt install curl
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
$ \. "$HOME/.nvm/nvm.sh"
```

- Install Node.js:

```
$ nvm install 16
$ nvm use 16
```

- Check install successful with:

```
$ node -v
$ npm -v
```

- Install gcc and g++ compilers (required to build some pip packages below)

```
$ sudo apt install gcc
$ sudo apt install g++
```

- Install Python package dependencies:

```
$ pip install -r requirements.txt
```

- Install and set up PostgreSQL connection:

```
$ sudo apt install postgresql
```

Create a file at path: test_server/backend/.env with these contents:

```
DB_NAME=basketify
DB_USER=basketify_user
DB_PWD=basketify1234
DB_HOST=database-1.ctee0c66o75x.us-west-1.rds.amazonaws.com
DB_PORT=5432
```

- Start the Django server on port 8000 & React server on port 300:

```
$ chmod +x start_server.sh
$ ./start_server.sh
```

Test Suite

Backend Tests

See test_server/backend/home/tests.py. Note that backend tests don't cover all FRs due to the nature of how the FRs are written and which data handling is done by the frontend vs the backend.
Note that FR7 and FR16 are rolled into the same feature (search for player is the same as prompt user for player to make ML predictions for), similarly for FR8 and FR17.

Test name	**Description**	**FR Coverage**	**Input**	**Expected Output**	**Actual Output**
test_welcome	Test the welcome message (used for debugging of Django connection) is correctly being passed	N/A (used for debugging)	GET request to `/`	HTTP response with 200 status code and JSON response: {"message": "Welcome to Django with react!"}	Same as expected
test_search_player_no_name	Test the search player route given a non-existent name field	FR7/FR16	GET request to `/search-player/` with no name parameter	HTTP response with 400 status code and error message	Same as expected
test_search_player_valid	Test the search player route given a valid player name parameter	FR7/FR16	GET request to `/search-player/?name=LeBron`	HTTP response with 200 status code and message with 2 players	Same as expected
test_search_player_no_results	Test the search player route given no matching results in DB	FR7/FR16	GET request to `/search-player/?name=NonExistentPlayer`	HTTP response with 404 status code and error message	Same as expected

test_search_team_no_name	Test the search team route with no name passed	FR8/FR17	GET request to `/search-team/`	HTTP response with 400 status code and error message	Same as expected
test_search_team_valid	Test the search team route with valid team name param	FR8/FR17	GET request to `/search-team/?name=Lakers`	HTTP response with 200 status code and team name in message	Same as expected
test_search_team_no_results	Test the search team route given no matching results in DB	FR8/FR17	GET request to `/search-team/?name=NonExistentTeam`	HTTP response with 404 status code and error message	Same as expected
test_get_player_stats_player_not_found	Test the get player stats route given no matching player in DB	FR9	GET request to `/stats/player/DoesNotExist/`	HTTP response with 404 status code and error message	Same as expected
test_get_player_stats_valid	Test the get player stats route given a valid player name	FR9	GET request to `/stats/player/LeBron%20James/`	HTTP response with 200 status code and complete stats object in “stats” field	Same as expected
test_get_team_stats_team_not_found	Test the get team stats route given no matching team in DB	FR9	GET request to `/stats/team/NoTeam/`	HTTP response with 404 status code and error message	Same as expected
test_get_team_stats_valid	Test the get team stats route given a valid team name	FR9	GET request to `/stats/team/Lakers/`	HTTP response with 200 status code and complete	Same as expected

				stats object in “stats” field	
test_predict_nba_champion	Test the predict season champion route given a good DB response	FR21	GET request to `/predict-season-champion/`	HTTP response with 200 status code and valid “top_team” and “top_team_ppg” fields	Same as expected
test_predict_nba_champion_no_data	Test the predict season champion route given an error response from DB	FR21	GET request to `/predict-season-champion/`	HTTP response with 500 status code and error message	Same as expected
test_register_user_success	Test successful user registration with valid data	FR1	POST request to `/api/register/` with valid email and password	HTTP 201 response with created user data	Same as expected
test_register_existing_email	Test registration with an email that already exists	FR1	POST request to `/api/register/` with an email already in use	HTTP 400 response with error message about duplicate email	Same as expected
test_register_invalid_email	Test registration with an invalid email format	FR1	POST request to `/api/register/` with improperly formatted email	HTTP 400 response with validation error	Same as expected
test_login_verified_user_success	Test successful login for verified user	FR2	POST request to `/api/token`	HTTP 200 response with access	Same as expected

			/ with valid credentials	and refresh tokens	
test_login_unverified_user	Test login attempt with unverified email	FR2	POST request to `/api/token/` with unverified user credentials	HTTP 401 response with 'email_not_verified' code	Same as expected
test_login_wrong_password	Test login attempt with incorrect password	FR2	POST request to `/api/token/` with wrong password	HTTP 401 response	Same as expected
test_email_change_request_success	Test successful email change request	FR3	POST request to `/accounts/email-change/` with valid user token	HTTP 200 response with success message	Same as expected
test_password_reset_request_success	Test successful password reset request	FR3	POST request to `/accounts/password-reset/` with valid email	HTTP 200 response with success message	Same as expected
test_verification_token_expiration	Test that verification tokens expire after 2 minutes	FR1	Set token creation time to 3 minutes ago	Token should be considered expired	Same as expected
test_login_non_existent_user	Test login attempt with email not in system	FR2	POST request to `/api/token` with non-existent email	HTTP 401 response	Same as expected
test_filter_params_are_processed	Test that filter parameters are correctly	FR25	GET request to player stats with	HTTP 200 response with filtered data	Same as expected

	extracted from request		filter parameters		
test_game_location_filter	Test filtering by game location (home vs away)	FR25, FR26	Game data with location filter	Only home or away games returned as specified	Same as expected
test_last_n_games_filter	Test filtering by last N games	FR25, FR26	Game data with last_n_games filter	Only the most recent N games returned	Same as expected
test_date_range_filter	Test filtering by date range	FR25, FR26	Game data with date_from and date_to filters	Only games within date range returned	Same as expected
test_opponent_filter	Test filtering by opponent team	FR25, FR26	Game data with opponents filter	Only games against specified team returned	Same as expected
test_outcome_filter	Test filtering by game outcome (Win/Loss)	FR25, FR26	Game data with outcome filter	Only games with specified outcome returned	Same as expected
test_season_type_filter	Test filtering by season type	FR25, FR26	Game data with season_type filter	Only games of specified season type returned	Same as expected
test_combined_filters	Test complex combination of multiple filter types	FR25, FR26, FR27	Game data with multiple filters	Only games matching all criteria returned	Same as expected
test_filter_reset	Test resetting all filters	FR28	Empty filter dictionary passed to filter function	All original games returned without filtering	Same as expected
test_season_year_filter	Test filtering by specific season year	FR25, FR26	Game data with season filter parameter	Only games from specified	Same as expected

				season returned	
test_conference_filter	Test filtering by conference (East/West)	FR25, FR26	Game data with conference filter parameter	Only games against teams from specified conference returned	Same as expected
test_division_filter	Test filtering by division (Atlantic, Central, etc.)	FR25, FR26	Game data with division filter parameter	Only games against teams from specified division returned	Same as expected
test_conference_type_filter	Test filtering by conference type (interconference/intraconference)	FR25, FR26	Game data with game_type filter parameter	Only inter/intra-conference games returned as specified	Same as expected
test_month_filter	Test filtering by specific month	FR25, FR26	Game data with month filter parameter	Only games from specified month returned	Same as expected
test_apply_filters_function	Test that the apply_filters_to_games utility function works correctly	FR25, FR26, FR27, FR28	Sample game data with various filters	Correctly filtered subset of games returned	Same as expected

ML Model Unit Tests

See test_server/pull_data_scripts/ml

Test name	**Description**	**FR Coverage**	**Input**	**Expected Output**	**Actual Output**
test_get_mongo_client	Test whether	N/A	N/A	MongoClient Instance	Same as expected

	MongoClient gets connected successfully				
test_get_mongo_client_failure	Test whether MongoClient fails return none	N/A	N/A	None	Same as expected
test_get_game_stats_success	Test game stats retrieval when data is found	N/A	Valid player/team name	List of recent games	Same as expected
test_get_game_stats_no_data	Test game stats retrieval when no data is returned	N/A	Invalid player/team name	Empty list	Same as expected
test_predict_next_game_vs_team_valid	Test prediction with CI returns valid tuple	FR18	Valid player + game history	Tuple (points, confidence)	Same as expected
test_predict_next_game_vs_team_invalid	Test prediction when insufficient history exists	FR18	Valid play + no game history	(None, None)	Same as expected
test_team_ppg_calc	Test average PPG for valid team data	FR21 Helper	Team with game data	Correct PPG Value	Same as expected
test_team_ppg_no_points	Test PPG function when no data exists	FR21 Helper	Team with missing points	None	Same as expected
test_predict_nba_champion	Test champion prediction	FR21	Teams with avg_ppg	Team with highest avg_ppg	Same as expected

	with valid team				
test_predict_nba_champion_no_data	Test champion prediction when no data is available	FR21	No team has avg_ppg	“No team has avg_ppg recorded”, 0	Same as expected
test_determine_win_loss_logic	Test win/loss function when team wins	FR20	Team score > opponent	W	Same as expected
test_determine_win_loss_loss	Test win/loss function when team losses	FR20	Team score < opponent	L	Same as expected
test_determine_win_loss_tie	Test win/loss function when teams tie	FR20	Team score = opponent	T	Same as expected
test_determine_win_loss_team_not_found	Test win/loss function when team isn’t found	FR20	Invalid Team names	Error message	Same as expected
test_determine_win_loss_game_not_found	Test win/loss function when game isn’t found	FR20	Game not found	Error message	Same as expected
test_store_feedback	Ensure store_feedback stores games AFTER last run only	FR23	Games before and after cutoff	Only future games stored	Same as expected
test_evaluate_feedback_discrepancies	Test large error and adjust accuracy	FR23, FR24	100 predicted, 60 actual	Slider -0.5 and report logged	Same as expected

test_evaluate_feedback_medium_error	Test medium error and adjust accuracy	FR23, FR24	25 predicted, 32 actual	Slider 0.25 and report logged	Same as expected
test_evaluate_feedback_no_points_field	Test missing data does not break logic.	FR23, FR24	Missing ‘Points’ in data	No update, empty report	Same as expected

Automated Frontend Tests

See test_server/frontend/__tests__

Test name	**Description**	**FR Coverage**	**Input**	**Expected Output**	**Actual Output**
render_loading_stats	Test the loading screen displays when rendering StatsPage	N/A (styling)	Render the stats page, check before loading screen done	Loading screen is present (message displayed)	Same as expected
render_stats_table	Test the stats table is displayed after page is rendered	FR9	Render StatsPage with valid mocked response from backend	“AllGames” table is present, selected stats from backend response are displayed	Same as expected
seasonal_game_toggle	Test the game-by-game/seasonal toggle button works	FR10	Render StatsPage with mocked response from backend. Verify game-by-game is shown by default, click the button, verify seasonal	“All Games” table is rendered, after one click, the seasonal stats values are shown, after another click, the game-by-game stats are shown	Same as expected

			stats displayed, click again, verify game-by-game stats displayed		
render_stats_no_data	Test a message is displayed when no stats available for player	FR9	Render StatsPage with a mocked empty response from backend	“No stats available for this player” message is shown	Same as expected
date_filter	Test that applying a date filter works to filter out a game	FR26	Render StatsPage with 3 mocked games as backend response, click on date filter, ensure one game (game C) isn’t displayed anymore	Game C isn’t displayed after filter is applied	Same as expected. Note that the assertions weren’t seeming to work, although manual acceptance testing shows they should.
handle_error_stats_page	Test errors from backend on StatsPage are handled gracefully	FR9	Render StatsPage with a mocked error as response from backend	“No stats available for this player” message is displayed after rendering	Same as expected
render_search_page	Test rendering of search page has all components present	FR7/FR8/FR16/FR17	Render SearchInterface page	“Enter player name” search bar is displayed and “Search players” appears twice on page	Same as expected

search_input	Test user input to search bar is actually collected	FR7/FR8/FR16/FR17	Render SearchInterface and input text to search bar	User input to search bar appears as text on screen	Same as expected
search_loading_screen	Test loading message is displayed when search is being down	N/A (styling)	Render SearchInterface, input 'LeBron', and click on search button	"Loading" message appears while search is being done	Same as expected
search_player	Test matching search results are displayed	FR7/FR16	Render SearchInterface with mocked response of 'LeBron James' as backend response, input 'LeBron' to search and click search button	'LeBron James' is displayed in the search results	Same as expected
search_player_no_results	Test message is displayed when no results match search for player name	FR7/FR16	Render SearchInterface with empty mocked backend response, search for 'Unknown Player'.	"No results found" message is displayed	Same as expected
search_team	Test matching search results are displayed	FR8/FR17	Render SearchInterface with mocked response of 'Los Angeles Lakers' as backend response, input 'Lakers' to search and click search button	'Los Angeles Lakers' is displayed in the search results	Same as expected

search_team_no_results	Test message is displayed when no results match search for player name	FR8/FR17	Render SearchInterface with empty mocked backend response, search for ‘Unknown Team’.	“No results found” message is displayed	Same as expected
search_toggle_player_team	Test the toggle button between player and team search works	FR7/FR8/FR16/FR17	Render SearchInterface, click on team search button, click on player search button	Initially we search for players, then search for teams, then search for players again	Same as expected
search_to_stats_navigation_player	Test that clicking on a search result navigates user to the stats page for that player	N/A (page navigation)	Render SearchInterface with mocked ‘LeBron James’ response from backend, search for ‘LeBron’, click on LeBron’s name	useNavigate should’ve been called with argument ‘/stats/player/LeBron James’	Same as expected
set_favourite_player	Test that clicking on player name while selecting favourite sends POST request to update favourite player	FR5	Render SearchInterface with ‘setFavorite’ attr, search for LeBron (mocked response has LeBron in results), and click on name	POST request sent to accounts/set-favorite/?type=player;name=LeBron%20 James	Same as expected
set_favourite_player_error	Test that clicking on player name while selecting	FR5	Render SearchInterface with ‘setFavorite’ attr, search	POST request sent to accounts/set-favorite/?type	Same as expected

	favourite sends POST request to update favourite player		for LeBron (mocked response has LeBron in results), and click on name	=player;name =LeBron%20 James	
set_favourite _team	Test that clicking on team name while selecting favourite sends POST request to update favourite team	FR6	Render SearchInterfa ce with ‘setFavorite’ attr, search for Oklahoma City Thunder (mocked response has OKC in results), and click on name	POST request sent to accounts/set-favorite/?type =team;name =Oklahoma% 20City%20Th under	Same as expected
set_favourite _team_error	Test that clicking on team name while selecting favourite sends POST request to update favourite team	FR6	Render SearchInterfa ce with ‘setFavorite’ attr, search for Oklahoma City Thunder (mocked response has OKC in results), and click on name	POST request sent to accounts/set-favorite/?type =team;name =Oklahoma% 20City%20Th under	Same as expected
ml_pred_load	Test loading screen is displayed while page is rendering	N/A (styling)	Render MLPrediction s page	“Loading” message is displayed	Same as expected
ml_pred_pre diction	Test the NBA champion predictions displays given valid backend response	FR22	Render MLPrediction s page with mocked backend response with top_team as “GSW” and “top_team_p pg” as 118.3	Predicted NBA champion displayed as Golden State Warriors in text and ppg matches backend response	Same as expected

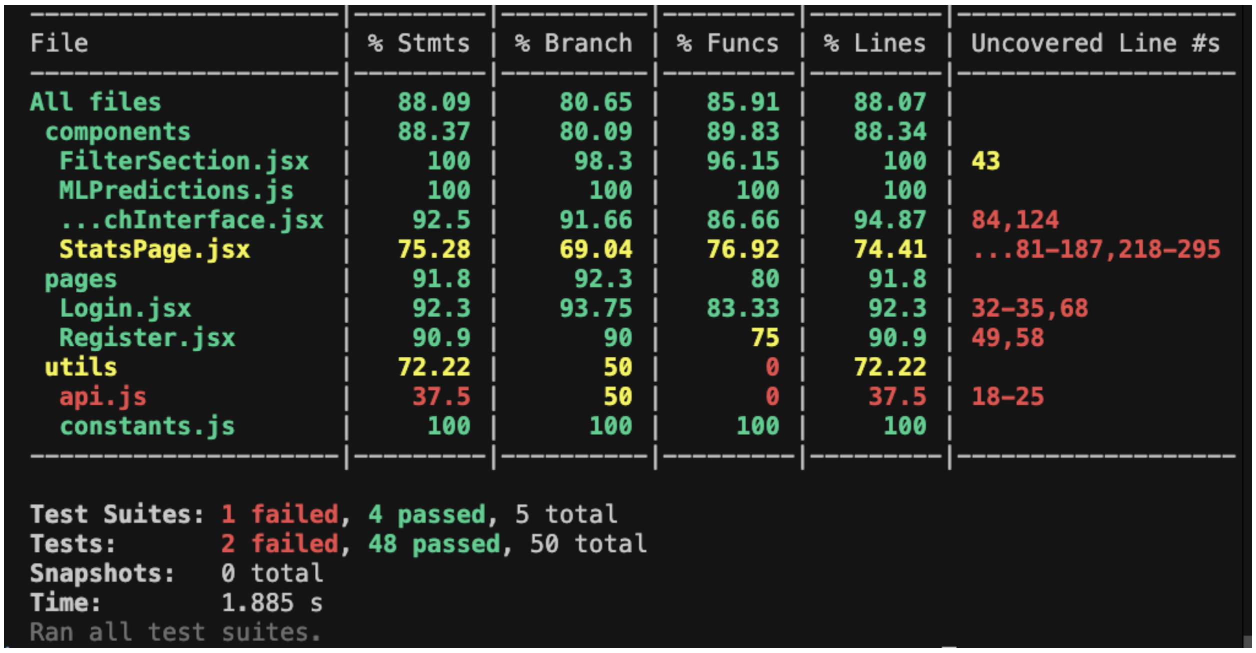
ml_pred_error	Test gracefully handling of backend errors	FR22	Render MLPredictions page with mocked error response from backend	Console logs error as “There was an error fetching the predicted NBA champion”	Same as expected
ml_pred_back_button	Test clicking on back button takes user back to dashboard	N/A (page navigation)	Render MLPredictions page and click on back button	useNavigate called with “-1”	Same as expected
render_registration_form	Test that registration form renders correctly	FR1	Render the Register component	Form with email, password fields, and register button visible	Same as expected
register_successful	Test successful registration submission	FR1	Fill and submit registration form with valid data	API called with correct data, success message displayed	Same as expected
register_error	Test registration error handling	FR1	Submit with email that already exists	Error message displayed to user	Same as expected
render_login_form	Test that login form renders correctly	FR2	Render the Login component	Form with email, password fields, and login button visible	Same as expected
login_successful	Test successful login submission	FR2	Fill and submit login form with valid credentials	API called with correct data, tokens stored	Same as expected
login_unverified_email	Test login with unverified email	FR2	Login with unverified email credentials	Display verification prompt and resend option	Same as expected

login_wrong_credentials	Test login with incorrect credentials	FR2	Login with incorrect email/password	Error message displayed	Same as expected
resend_verification_email	Test resending verification email	FR1, FR2	Click resend verification button	API called to resend email	Same as expected
render_password_reset_link	Test password reset link visibility	FR3	Render the Login component	Password reset link is visible	Same as expected
render_email_change_link	Test email change link visibility	FR3	Render the Login component	Email change link is visible	Same as expected
filter_section_not_visible_when_closed	Test filter section hidden state	FR25	Render FilterSection with isOpen=false	Filter options not visible in DOM	Same as expected
filter_section_visible_when_open	Test filter section visible state	FR25	Render FilterSection with isOpen=true	Filter options visible in DOM	Same as expected
filter_displays_all_criteria_options	Test all filter options are displayed	FR25	Render FilterSection with isOpen=true	All filter categories are visible	Same as expected
filter_date_range_update	Test date range filter updates	FR26	Change date inputs and apply	Callback called with correct date range values	Same as expected
filter_last_n_games_update	Test last N games filter updates	FR26	Change last N games input and apply	Callback called with correct last_n_games value	Same as expected
filter_season_update	Test season filter updates	FR26	Select a specific season and apply	Callback called with correct season value	Same as expected

apply_multiple_filters_simultaneously	Test applying multiple filters	FR27	Set multiple filter values and apply	Callback called with all selected filter values	Same as expected
opponents_filter_multiple_selection	Test opponent multi-select	FR27	Select multiple opponents and apply	Callback called with correct opponents string	Same as expected
clear_filters_button_resets_all_filters	Test clear filters functionality	FR28	Click clear filters button	All filter inputs reset, callback called with empty object	Same as expected
initialize_with_existing_filters	Test initializing with existing filters	FR25, FR26	Render with initialFilters prop	Filter inputs show initial values	Same as expected
removing_opponent_from_selection	Test removing selected opponents	FR27	Click remove button for a selected opponent	Opponent removed from selection	Same as expected
filter_season_type_update	Test filtering by season type (Regular Season/Playoffs)	FR25, FR26	Select a specific season type and apply	Callback called with correct season_type value	Same as expected
filter_division_update	Test filtering by division	FR25, FR26	Select a specific division and apply	Callback called with correct division value	Same as expected
filter_conference_update	Test filtering by conference	FR25, FR26	Select a specific conference and apply	Callback called with correct conference value	Same as expected
filter_game_type_update	Test filtering by game type (Interconference/Intraconference)	FR25, FR26	Select a specific game type and apply	Callback called with correct game_type value	Same as expected

filter_month_update	Test filtering by month	FR25, FR26	Select a specific month and apply	Callback called with correct month value	Same as expected
multiple_dropdown_filters	Test applying multiple dropdown filter types	FR27	Select values for multiple dropdowns and apply	Callback called with all selected dropdown values	Same as expected

Frontend coverage report:



```
----------------------|---------|----------|---------|---------|-------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------
All files             |   88.09 |    80.65 |   85.91 |   88.07 |
 components           |   88.37 |    80.09 |   89.83 |   88.34 |
  FilterSection.jsx   |     100 |     98.3 |   96.15 |     100 | 43
  MLPredictions.js    |     100 |      100 |     100 |     100 |
  ...chInterface.jsx  |    92.5 |    91.66 |   86.66 |   94.87 | 84,124
  StatsPage.jsx       |   75.28 |    69.04 |   76.92 |   74.41 | ...81-187,218-295
 pages                |    91.8 |     92.3 |      80 |    91.8 |
  Login.jsx           |    92.3 |    93.75 |   83.33 |    92.3 | 32-35,68
  Register.jsx        |    90.9 |       90 |      75 |    90.9 | 49,58
 utils                |   72.22 |       50 |       0 |   72.22 |
  api.js              |    37.5 |       50 |       0 |    37.5 | 18-25
  constants.js        |     100 |      100 |     100 |     100 |
----------------------|---------|----------|---------|---------|-------------------

Test Suites: 1 failed, 4 passed, 5 total
Tests:       2 failed, 48 passed, 50 total
Snapshots:   0 total
Time:        1.885 s
Ran all test suites.
```

System/Acceptance-level Tests

Test name	**Description**	**FR Coverage**	**User Actions**	**Expected Results**	**Actual Results**
graph_render_one_stat	Tests that a stat can be selected and the graph is updated	FR11/FR12/FR14/FR18 FR14 = hoverable FR18 = future game	From the dashboard, search for ‘LeBron James’, click on his name, click on the graph of stats	A graph of points per game this season is displayed, with future predictions shown as	Same as expected

		predictions	button, then click on Points	yellow stars. Hovering over stats shows exact values	
graph_render_two_stats	Test that 2 stats can be displayed on the graph at once	FR11/FR12/FR13	From the previous test: click on Assist, then de-select Assists	First click on Assists shows 2 series on same graph, de-selecting Assists removes it from the graph	Same as expected
graph_max_one_stat_selectable	Test that max of 2 stats are selectable	FR12	Select Points, Assists, and Rebounds	Clicking on Rebounds does nothing (3rd stat),	Same as expected
graph_toggle_game_by_game_seasonal	Test that game-by-game and seasonal toggle button works	FR15	From the stats graph, with Points selected, hit the seasonal stats button	Graph changes to display aggregated points on seasonal basis back to 2009-2010 season	Same as expected
dashboard_search_connection	Test the search connection button navigates to search page	FR4	From the dashboard, click on Search Player/Team button	User is taken to SearchInterface page	Same as expected
dashboard_favourite_player_connection	Test favoriting of player	FR5	From the dashboard with a new user account, select Favourite Player, search for LeBron James, select his name,	After selecting favourite player, dashboard has LeBron James text and clicking on button takes user to stats page for	Same as expected

			then once back at dashboard (taken back automatically), click on Favourite Player	LeBron James	
dashboard_favourite_team_connection	Test favoriting of team	FR6	From the dashboard with a new user account, select Favourite Team, search for Lakers, select the team, then once back at dashboard (taken back automatically), click on Favourite Team	After selecting favourite team, dashboard has Los Angeles Lakers text and clicking on button takes user to stats page for Lakers	Same as expected
user_registration_complete	Full user registration and verification workflow	FR1	1. Navigate to registration page 2. Enter email and password 3. Submit form 4. Check verification email 5. Click verification link	1. Form submits successfully 2. Success message displayed 3. Email received 4. Clicking link verifies account 5. Can log in with credentials	Same as expected
login_unverified_resend_verify	Login with unverified account and resend	FR1, FR2	1. Register account but don't verify	1. Login fails with verification error	Same as expected

	verification		2. Attempt to login 3. Click resend verification 4. Verify with new link	2. Resend option appears 3. New email received 4. Account verified successfully	
password_reset	Full password reset workflow	FR3	1. Navigate to password reset form 2. Enter email 3. Check email 4. Click reset link 5. Set new password	1. Reset request confirmed 2. Email received 3. Reset page loads 4. Password changed 5. Can login with new password	Same as expected
email_change	Full email change workflow	FR3	1. Login 2. Navigate to email change 3. Request change 4. Confirm with links 5. Verify new email	1. Change request confirmed 2. Emails sent to both addresses 3. New email verified 4. Login works with new email	Same as expected
filter_stats_date_range	Filter statistics by date range	FR25, FR26	1. Navigate to stats page by searching for player/team and click on	1. Filter button is visible on stats page 2. Can view	Same as expected

			that player/team to view their stats 2. Open filter section by clicking on the “Filter” button 3. Set date range filters 4. Apply filters by clicking on “Apply Filters” in filter section	the list of available filters to apply 3. After applying, only games within range are displayed 4. Filter status shows active filters which include date range and games that only satisfy the date range are included	
filter_stats_multiple_criteria	Filter statistics with multiple criteria	FR25, FR26, FR27	1. In filter section, select multiple filter criteria(e.g. Opponent and last 5 games) 2. Apply filters by clicking on “Apply Filters” in filter section	Filter status shows active filters which include opponent and last 5 games and games that only satisfy the opponents selected and are within the last N games are included	Same as expected
filter_stats_clear_filters	Test clearing all filters	FR28	1. Apply several filters 2. Click "Clear Filters” 3. Apply empty filters	1. All filter inputs reset 2. Filter status disappears 3. All games are displayed again	Same as expected