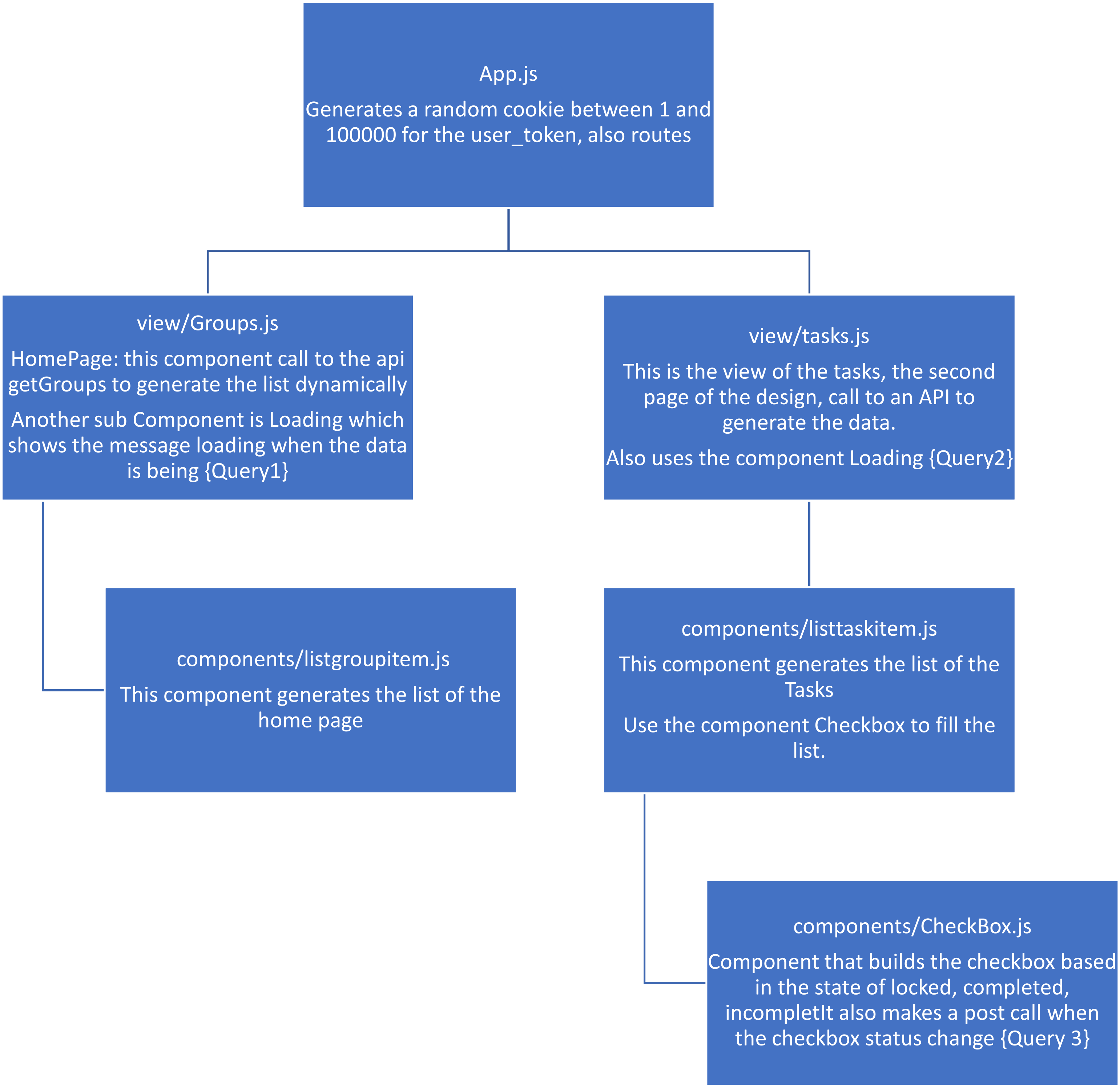
MODULES ARCHITECTURE



App.js

Generates a random cookie between 1 and 100000 for the user_token, also routes

view/Groups.js

HomePage: this component call to the api getGroups to generate the list dynamically

Another sub Component is Loading which shows the message loading when the data is being {Query1}

view/tasks.js

This is the view of the tasks, the second page of the design, call to an API to generate the data.

Also uses the component Loading {Query2}

components/listgroupitem.js

This component generates the list of the home page

components/listtaskitem.js

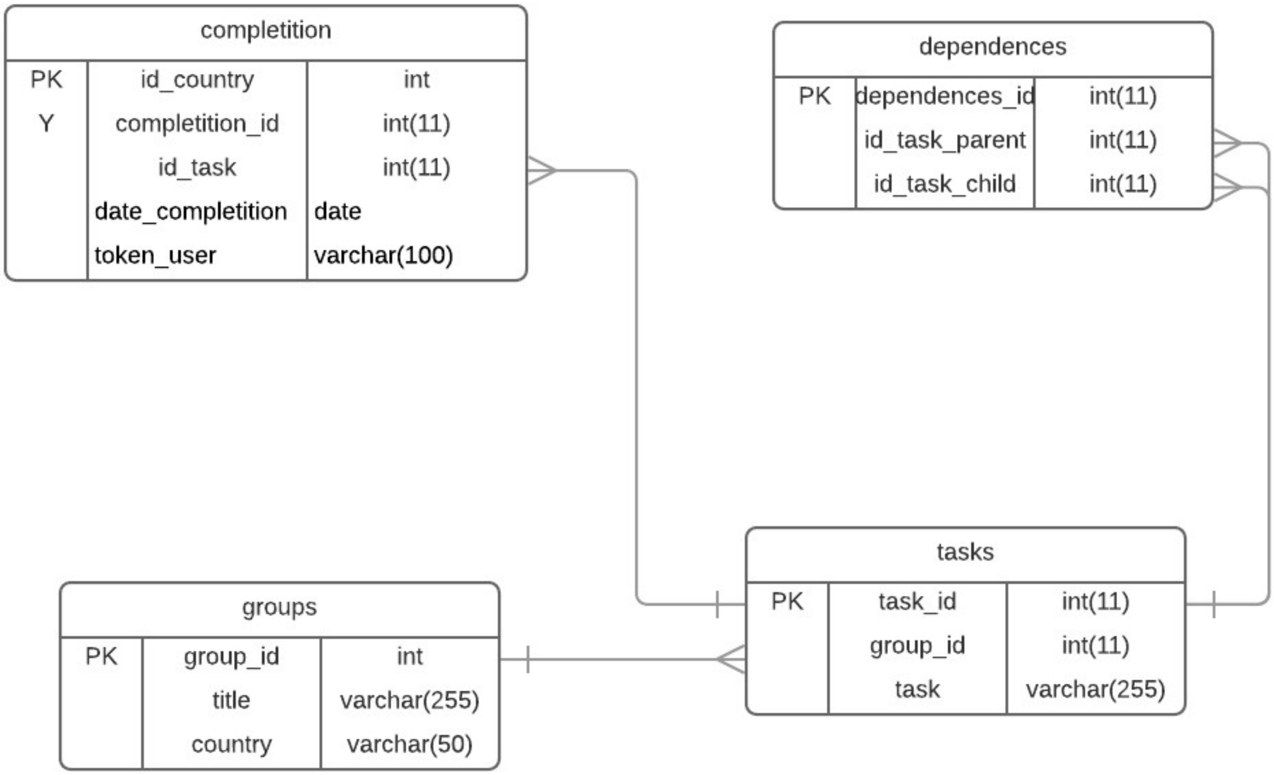
This component generates the list of the Tasks

Use the component Checkbox to fill the list.

components/CheckBox.js

Component that builds the checkbox based in the state of locked, completed, incompletIt also makes a post call when the checkbox status change {Query 3}

DATABASE ARCHITECTURE



completition

PK	id_country	int
Y	completition_id	int(11)
	id_task	int(11)
	date_completition	date
	token_user	varchar(100)

dependences

PK	dependences_id	int(11)
	id_task_parent	int(11)
	id_task_child	int(11)

tasks

PK	task_id	int(11)
	group_id	int(11)
	task	varchar(255)

groups

PK	group_id	int
	title	varchar(255)
	country	varchar(50)

The architecture was designed in the easiest was possible

The table groups have information related to the groups.

The table tasks have information related to the tasks.

Table dependences is a table with a tree structure. In which the dependencies are detailed.

The table competition is a table with information about when a user completes a task.

PD: The database can be generated using the file data_dump.dmp

SERVER SIDE

The server side is configured in express, more information can be found in the GitHub repository or upon request, two files had been generated, the file inside the folder api have the logic, the file inside the folder domain have the SQL query:

- backend-reactjsdemo/api/dependencies.js
- backend-reactjsdemo/domain/dependencies.js

{Query1}: This Query get all the groups and the tasks per user

```
SELECT g.group_id,
       g.title,
       COUNT(1) total,

  (SELECT COUNT(1)
   FROM completition c3
   JOIN tasks t2 ON (t2.task_id = c3.id_task)
   WHERE c3.token_user = ${user_token}
     AND t2.group_id = g.group_id) completed
FROM groups g,
     tasks t
WHERE g.group_id = t.group_id
GROUP BY g.group_id
```

{Query2}: This Query get all the tasks, the logic behind it is if is_Completed > 0 then the task is completed, if the difference between dependents and number of completed is bigger than 0 then the task is locked and if the difference is 0 and is_completed == 0 then the task is available and unlocked.

```
SELECT task_id,
       task,
       dependents,
       number_completed,
       dependents - number_completed difference,
       is_completed
FROM
  (SELECT t.task_id,
          t.task,
          Count(d.id_task_child) dependents,

     (SELECT Count(1)
      FROM completition c2
      JOIN dependences d2 ON(c2.id_task = d2.id_task_child)
      WHERE d2.id_task_parent = t.task_id
        AND c2.token_user = ${user_token}) number_completed,

     (SELECT COUNT(1)
      FROM completition c3
      WHERE c3.id_task = t.task_id
        AND c3.token_user = ${user_token}) is_completed
   FROM tasks t
   LEFT JOIN dependences d ON(t.task_id = d.id_task_parent)
   WHERE t.group_id = ${group_id}
   GROUP BY t.task_id) child_table
```

{Query2}: When an user marks a task as completed, then this api is called which inserts data in the database.

```
INSERT INTO completition (id_task, date_completition, token_user)
VALUES(${task_id}, NOW(), ${user_token})
```