

UNIVERSIDAD AUTÓNOMA DE MADRID

DEPARTAMENTO DE INFORMÁTICA

---

# Computer Systems Project

## Assignment 4

---

Roberto MARABINI  
Alejandro BELLOGÍN

## Changelog

| Version <sup>1</sup> | Date       | Author | Description                           |
|----------------------|------------|--------|---------------------------------------|
| 1.0                  | 10.08.2022 | RM     | First version.                        |
| 2.0                  | 26.10.2022 | RM     | Change heroku → render.com.           |
| 2.1                  | 5.12.2022  | RM     | Renumbering assignment 5 → 4          |
| 2.2                  | 22.12.2022 | AB     | Translation to English                |
| 2.3                  | 29.1.2023  | RM     | Review before upload to <i>Moodle</i> |

---

<sup>1</sup>Version control is made using 2 numbers  $X.Y$ . Changes in  $Y$  denote clarifications, more detailed descriptions of some aspect, or translations. Changes in  $X$  denote deeper modifications that either change the provided material or the content of the assignment.

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Goals</b>   | <b>3</b>  |
| <b>2</b> | <b>Work to be done during the first half of the assignment</b> | <b>3</b>  |
| 2.1      | Implementing a REST API . . . . .                              | 3         |
| <b>3</b> | <b>Work to be developed during the rest of the assignment</b>  | <b>5</b>  |
| 3.1      | Implementation . . . . .                                       | 6         |
| 3.2      | git . . . . .  | 7         |
| <b>4</b> | <b>Work to present at the end of the assignment</b>            | <b>8</b>  |
| <b>5</b> | <b>Evaluation criteria</b>                                     | <b>9</b>  |
| <b>A</b> | <b>Images</b>  | <b>13</b> |

# 1 Goals

In this assignment, you will finish implementing the *kahootclone* application by developing the necessary methods for the participants to answer the questionnaires' questions. We remind you in this assignment we WILL value the aesthetics of the web application together with its usability. Both the aspect and design of the application will remain your choice, although we request you NOT to be visually similar to the one provided as reference.

## 2 Work to be done during the first half of the assignment

The work to be done starts from the code developed during the previous assignment, so it is recommended you keep on using the same private repository BUT creating another **branch** called `assignment_4`, to isolate the code for each assignment. At the same time, whenever you deploy the application in *render.com*, create a project and a new database, not overwriting the one used in the previous assignment, since it will be necessary to evaluate the previous assignment.

In the previous assignment, the *Django* environment was used to create the GUI of the application users; in this assignment we will use *Vue.js* to create the graphical interface that the participants will use instead of *Django*. *Vue.js* will have access to the database where the models are stored through a REST API that we will build using the **Django Rest Framework**. The calls will be made using the `fetch` module or the `Axios` module from *Vue.js*.

### 2.1 Implementing a REST API

You may reuse the *django* project created in the previous assignment and add it a new application called `restServer`, where all the code described next will be hosted. We will need three methods (you may see in part the result of implementing them in <https://kahoortclone.onrender.com/api/>). The first one will allow us to join the game. The second one will implement the waiting step until all the participants have

joined the game, and the third one will allow us to send the responses (**guess**) to the proposed questions. In the following, we describe these services in detail. Recall that these are REST services and, because of that, they receive and return a dictionary in JSON format. For each service, a URL is provided where they should listen, a/some available test(s) in the repository (see file `djangoRestServer/tests.py`) (the six first letters for each set of tests are provided) and a description including the expected “input” and “output”.

`/api/participant/` — `test02...` — It receives a dictionary `{'game': int, 'alias': string}` including the values of `game.publicId` and `participant.alias`. The method creates a participant (with the received alias and `game`) and returns the created participant, which contains the identifier `uuidP` that will be used in the future to identify the participant. Note that in REST APIs it is not possible to use session variables.

`/api/games/` — `test01...` — It receives a dictionary `{'publicId': int}` including a `game.publicId` and returns a serialized `game` object. It is useful to check when the state of the game (`game.state`) changes from `WAITING` to `QUESTION`, which occurs when the allocated time for participants to join has ended and the button to start the game has been pressed.

`/api/guess/` — `test03...` — It receives a dictionary `{'game': int, 'uuidp': string, 'answer': int}` with a `game.publicId`, a `participant.uuidP`, together with an integer number in the range `[0 – 3]` that stores the selected answer by the participant. With this information, a **guess** is created. The method admits the creation of a unique **guess** for each tuple (participant, question), at the same time it checks before adding the **guess** that the participant exists and that the state of the game (`game.state`) is `QUESTION`.

In all the cases, the most obvious errors need to be managed, for example:

- Selecting the same alias by two participants. In this case, an error message will be returned.

- Preventing that the same participant sends two answers to one question. Only the first answer will be admitted, and an error message will be shown in the second (or additional) attempt.
- Sending incorrect identifiers. For example, game identifiers that are already closed, participant identifiers not linked to open games, or that do not exist.

Note: By default the REST API of *Django* creates a set of methods with no access restrictions. It is your responsibility to take care that the data model database cannot be accessed for different functions to those described in this part. There are ways to achieve this goal, in the implementation we show in <https://kahoorclone.onrender.com> we managed a finer control of the access permission by redefining the method `get_permissions`. A negative effect of limiting the permissions is that the usefulness of the test interface <https://kahoorclone.onrender.com/api/> is reduced drastically, so it might be interesting to create a first version of the API without restrictions until the interface is working properly.

**Testing** To verify the correct implementation of the REST API, we have defined in file `djangoRestServer/tests.py` a set of tests (not necessarily complete) that your code must satisfy. These tests must be understood as additional requirements to the project.

### 3 Work to be developed during the rest of the assignment

During this second half of the assignment, you will implement the web pages that will consume the REST services, and you will deploy the application in *render.com*, as it was described in the previous assignment. Recall the deployment should be done in production mode. At the same time, you must write a user manual, which must provide instructions about how to use your web application. You must not describe how you have implemented it, which problems you found during the implementation,

or any other aspect related to the implementation. Just focus on the use of the application, together with creating and playing questionnaires.

The three methods described in the previous paragraph must be called by three pages created in *Vue.js* using *fetch* (see <https://kahootclone-render-vue.onrender.com>). In this paragraph, we use the term page freely, as we actually refer to three URLs defined by *router* from *Vue.js*. The first page will show a form asking the *participant.alias* and the *game.publicId*, and it will call to */api/participant* to create the new participant (Fig. 1). The second one will show a message to the participant telling them that they are waiting until the game starts (Fig 2). This page must check the state of the game *game.status* every 2 seconds calling to */api/games*, when *game.status* goes from *WAITING* to *QUESTION* it will move to the third page (Fig. 3). This last page will allow to choose among the four available answers (a number in the  $[0 - 3]$  range) and will send the choice to the server by calling to */api/guess*. After sending an answer, a message must be received confirming if the submission was done successfully or not.

### 3.1 Implementation

Create a project called *vueClient* with the command `npm init vue@3.2` (alternatively, you may use `make init` using the existing makefile in the folder *vueClient*). This command initializes the version 3.2 of *vue* using the *vite* utility. Choose the following options (those marked with two asterisks, that is, *\*XX\**):

```
# Project name: ... vueClient
# Add TypeScript? ... *No* / Yes
# Add JSX Support? ... *No* / Yes
# Add Vue Router for Single Page Application development? ... No / *Yes*
# Add Pinia for state management? ... *No* / Yes
# Add Vitest for Unit Testing? ... *No* / Yes
# Add Cypress for both Unit and End-to-End testing? ... *No* / Yes
# Add ESLint for code quality? ... No / *Yes*
# Add Prettier for code formatting? ... No / *Yes*
```

and run

```
cd vueClient
npm install
make requirements
```

These commands will install the following modules

```
bootstrap@5.1.3
@popperjs/core@2.11.5
axios@0.27.2
vuex@4.0.2
```

Recall that `npm run dev` (or `make runserver`) starts the server and `npm run lint` (or `make lint`) runs the `lint` program. `axios` is a library similar to `fetch`, `vuex` is used to create session variables, and `bootstrap` gives access to the CSS framework `bootstrap`. You are not forced to use these libraries but you may if you find it convenient.

Next, implement the three requested pages being the “homepage” the one that shows a form asking for the `participant.alias` and the `game.publicId`.

### 3.2 git

As usual, save and share your code using *git*. `make init` and `make install` are going to install a collection of modules in the folder `node_modules`. Do not upload this folder to the repository, as its content depends on your operating system.



## 4 Work to present at the end of the assignment

- Make sure your code implements all the required functionality and satisfies all the provided tests. It is not acceptable to modify the code of the tests except the variables located at the beginning of the file, which are clearly identified.
- Implement all the tests you consider necessary to reach full **coverage** of the developed functionality. These tests must be implemented in a file called `tests_additional.py`.
- Include in the root of the project a file called `coverage.txt` which contains the result of running the command **coverage** for all the tests.
- Include in the root of the project a file called `user_guide.pdf` containing the user manual.
- Deploy and test the application in *render.com* in production mode.
- This assignment requires to present 2 projects: one in *Django* that manages the administrative part and which contains the REST API, and one in *Vue.js* where the participants connect to. Upload to *Moodle* two files, obtained from running the command `zip -r ../xxxxx.zip .git` from the root of the *Django* and *Vue.js* projects respectively. Change `xxxxx` to *Django* or *Vue.js* depending on the project. Recall you must add and “commit” the files to git before running that command. If you want to check the content of the zip file is correct, you may do it by running the command: `cd ..; unzip assign5_final.zip; git clone . tmpDir; ls tmpDir`.
- Make sure the variable `ALLOWED_HOSTS` from file `settings.py` included in the submission contains the deployment path in *render.com*. Similarly, it is expected the variables **Cross-Origin Resource Sharing** (`CORS_ORIGIN_WHITELIST`) are properly initialized to be able to run the

application in *render.com*. Finally, check in *render.com* that both the user name and password for admin are *alumnodb*. If any of the two needed URLs to execute the assignment in *render.com* are not available as indicated in this point or the admin user password is not the proposed one, the assignment will be evaluated as if it was not deployed in *render.com*.

- Check that, when writing files in Python, code presents a consistent style and according to the style criteria highlighted by *Flake8*. *Flake8* must not return any error when executed on the code programmed by the student.
- Similarly, the files created with *Vue.js* must satisfy the criteria highlighted by *lint* (command `npm lint`)

## 5 Evaluation criteria

To pass with 5 points it is necessary to satisfy the following criteria completely:

- It is possible to create a questionnaire and play with it locally. The score of the participants is shown after every question.
- The file uploaded to *Moodle* includes a git repository.
- The code has been stored in a git repository accesible to all the members of the work pair and this repository is private.
- When running the tests, the number of fails is not larger than ten, and the code that satisfies the tests is functional.

If the following criteria are accomplished, a grade up to 5.9 might be achieved:

- All the previous criteria are accomplished completely.
- When running the tests in local, the number of fails is not larger than six and the code that satisfies them is functional.

- It is not possible to create/delete/edit questionnaires/questions/answers for users NOT connected (“logged”) even when accessing directly to the associated URLs with the different services.
- It is not possible to join games, or create results (**guess**) for participants without knowing the **publicId** of a game, even when directly accessing the associated URLs with the different services.

If the following criteria are accomplished, a grade up to 6.9 might be achieved:

- All the previous criteria are accomplished completely.
- The two requested projects are deployed in *render.com*. In the file **settings.py** the path of the first project in *render.com* is assigned to the variable **ALLOWED\_HOSTS** and the path corresponding to the second one to **CORS\_ORIGIN\_WHITELIST**. Besides being deployed, both projects must work correctly in *render.com*.
- The database admin application (admin/) is deployed and accesible in *render.com* using the username/password *alumnodb*.
- The application is deployed in *render.com* in production mode.
- Using the admin application it is possible to create, list, or remove objects belonging to all the required models.

If the following criteria are accomplished, a grade up to 7.9 might be achieved:

- All the previous criteria are accomplished completely.
- Heritage has been used in *Django* when creating the web pages.
- Every form that is either related with a model, or requested explicitly in the assignment, has been built using the form system implemented by *Django*.
- CSS files or environments such as *bootstrap* have been used to define the styles.
- When running the tests, the number of fails is not larger than four and the code that satisfies them is functional.

- The web application has a user interface that is user friendly and visually neat. Visually the web application must NOT be similar to the model we have provided in *render.com*.

If the following criteria are accomplished, a grade up to 8.9 might be achieved:

- All the previous criteria are accomplished completely.
- The code is readable, efficient, well-structured, and commented.
- The tools provided by the framework are used.
- The following are examples of the previous points:
  - The queries are done by the database. That is, the methods in `views.py` do not retrieve all elements of a table (i.e. `ClassName.objects.all()`) and the search is done by the functions defined in `views.py` (for `object` in `class`: `if object.name=='Pedro': ...`).
  - The errors are properly processed and understandable message errors are returned.
  - The web application properly manages accesses to non existing pages or with inadequate permissions.
  - The code presents a consistent style and the functions are commented including their author. Note: the author of a function must be unique.
  - The style criteria highlighted by *Flake8* are applied in a coherent way. *Flake8* does not return any error when executed on the code programmed by the student.
  - The files created with *Vue.js* must satisfy the criteria highlighted by *lint* (command `npm lint`).
- The user manual is included and appropriate.
- When running the tests, the number of fails is not larger than two and the code that satisfies them is functional.

- If we reduce the size of the browser window or use the zoom, all the elements in the page are still accessible and no functionality is lost.

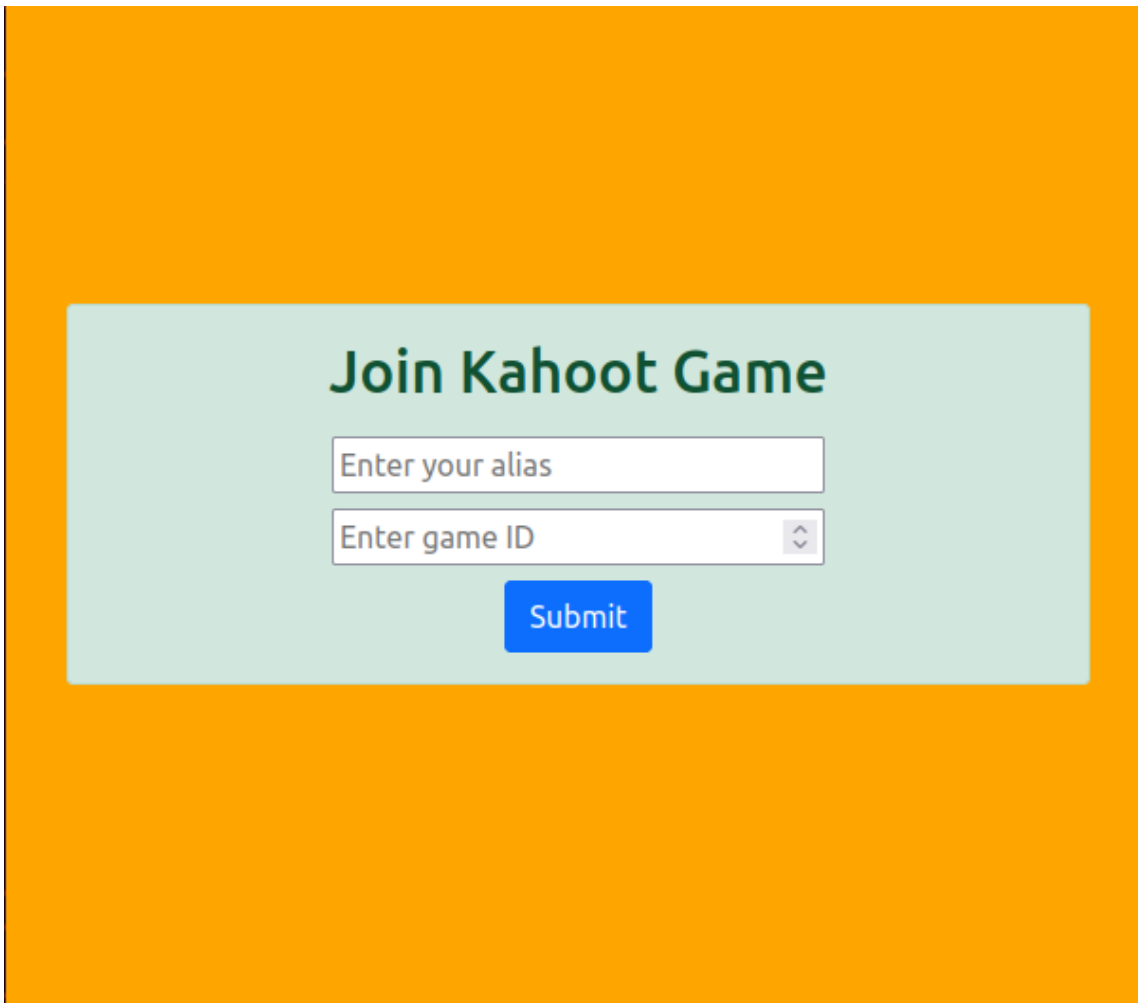
If the following criteria are accomplished, a grade up to 10 might be achieved

- All the previous criteria are accomplished completely.
- The application is robust and responds adequately even if invalid parameters are provided.
- All the tests provide satisfactory results.
- The coverage for the *Django* files that contain the models, views, and forms is over 99%. If the proposed tests do not generate the necessary coverage, create a new file called `tests_additional.py` and add the necessary tests in it.

Note: Late final submission → take away a point for each late day (or fraction) in the submission.

Note: The code used in the assessment of the assignment will be the one submitted to *Moodle*. Under no circumstance, the existing code in *render.com*, *Github*, or any other repository will be used.

## A Images

A screenshot of a web form titled "Join Kahoot Game" centered on a solid orange background. The form itself is a light green rounded rectangle. It contains a title "Join Kahoot Game" in a bold, dark green font. Below the title are two input fields: the first is a text box with the placeholder "Enter your alias", and the second is a text box with the placeholder "Enter game ID" and a small dropdown arrow on its right side. Below these two fields is a blue rectangular button with the word "Submit" in white text.

**Join Kahoot Game**

Enter your alias

Enter game ID

Submit

Figure 1: Page used for the participants to join a game.

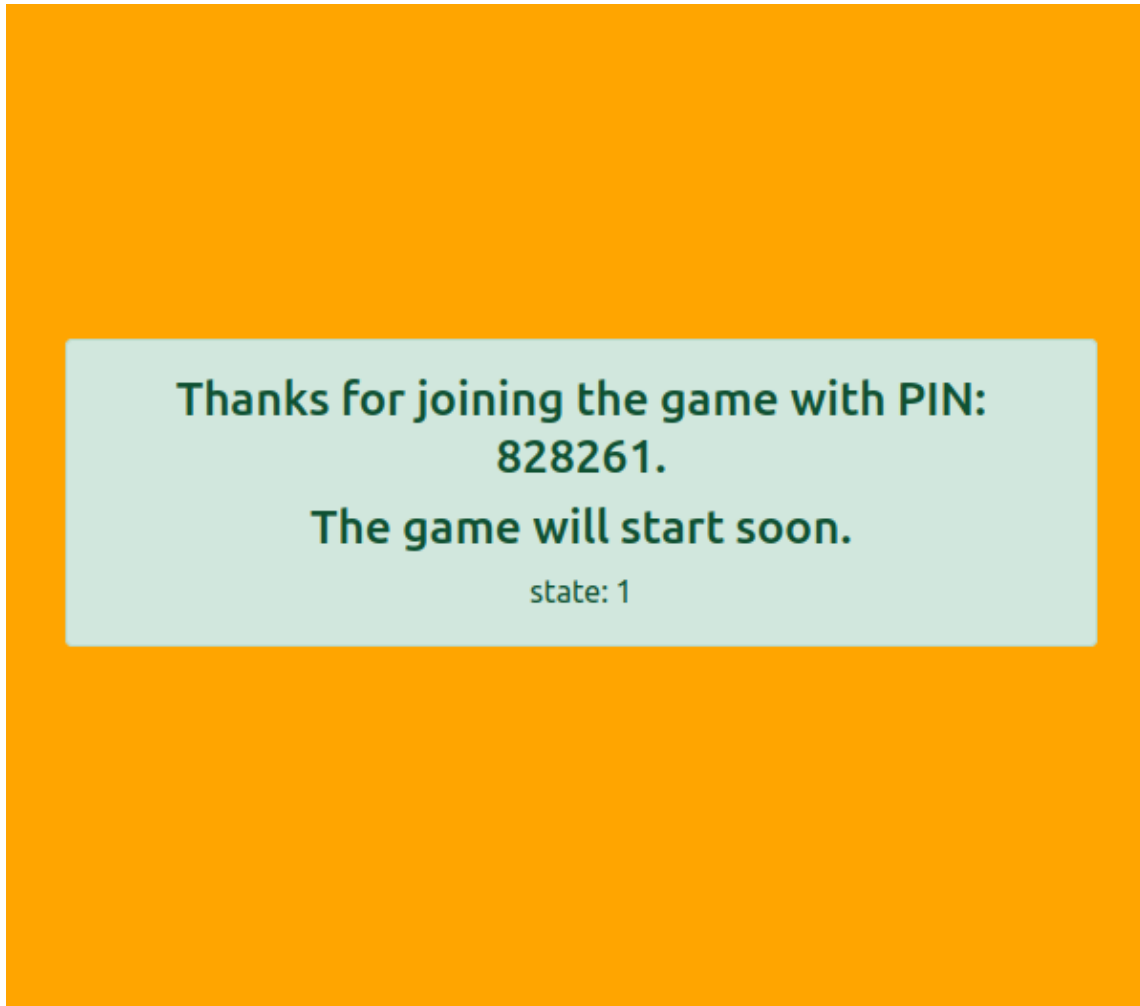


Figure 2: Waiting page until the game starts.

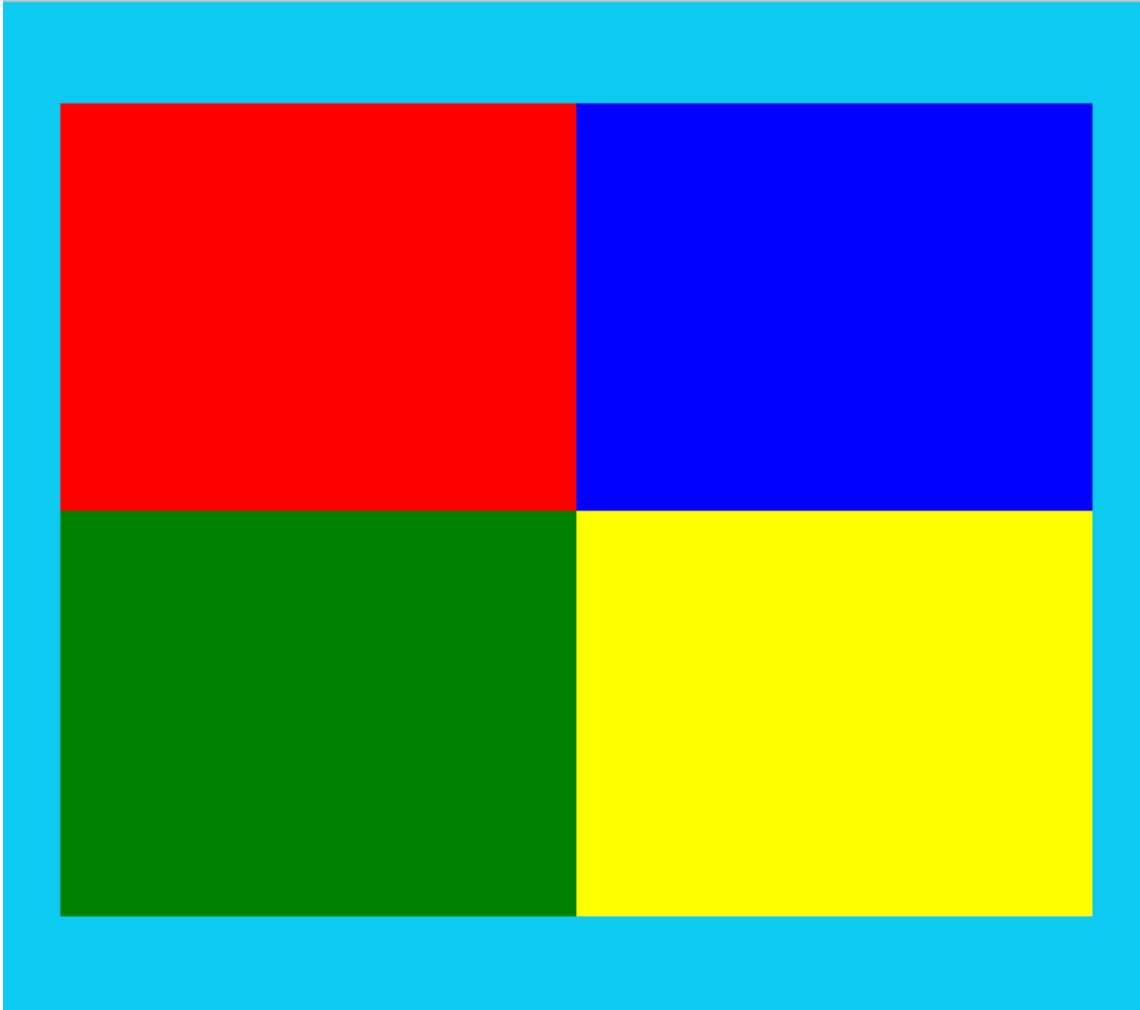


Figure 3: Page used to select an answer.