Estadísticas para conocer al año

Universidad de La Habana
Facultad de Matemática y Computación

Desarrolladores:
Olivia González Peña C-411
Sheyla Cruz Castro C-412
Laura Brito Guerrero C-412
Juan Carlos Casteleiro Wong C-411

1. Problema

Planteamiento del problema

Usuario final: Fernando Rodríguez Flores @fernan2rodriguez

Objetivos

El objetivo de este proyecto es diseñar e implementar un sistema (¿de encuestas? ¿de preguntas? ¿de tarjetas por el día de las madres?) que permita obtener información sobre un grupo de clases al comienzo del curso. Entre lo que se desea conocer están los intereses de los estudiantes (qué les gusta y a qué le dedican su tiempo libre). Cuánto conocen sobre determinados temas (videojuegos, películas, libros). Si juegan, leen o consumen series, y en esos casos qué tipos de juegos, libros, o series. Por ejemplo, en caso de que a alguien le guste la poesía, qué tipo de poesía, qué autores. ¿Qué tipo de música? Etc. ¿Cuál es su relación con las asignaturas que ya vieron? ¿Cuáles le gustaron? ¿Cuáles no les gustaron? ¿Cuáles les dio absolutamente lo mismo? ... etc.
El objetivo de esta primera parte es determinar qué tipo de elementos y referencias se pueden incluir en las conferencias y clases prácticas de forma que sean relevantes para la mayor cantidad de personas posibles.
También interesa conocer los niveles de creatividad y de sentido del humor que tengan los estudiantes. Esto es a través de los *tests* y *cuestionarios* ya establecidos para esos fines.
Algunas de estas informaciones conviene que sea *con nombre*, otras a lo mejor conviene que sea anónima.
Como parte del proyecto de debe:

1. Crear un mecanismo que permita recoger esta información *de la mejor manera posible*. *De la mejor manera posible* significa que quizás la mejor opción no sea ponerles una encuesta con varios cientos de incisos ... o a lo mejor sí.
2. Crear un sistema que permita analizar la información recogida.
3. Mostrar la información recopilada de una manera que resulte útil para la planificación de las asignaturas.

2. Solución

2.1 Back-End La solución desarrollada incluye la implementación de diversas estructuras de datos que permiten manejar la información más cómodamente. La lógica seguida para el algoritmo está basada en el análisis estadístico de las respuestas de los estudiantes a las encuestas, tanto de manera indvidual como general. A continuación, los detalles:

- **poll.py**: En este fichero se recogen en un diccionario los datos de la aplicación, es decir, los temas a considerar en las encuestas y las respuestas de los estudiantes a las mismas. Se tiene en consideración el anonimato de las encuestas debido a la especificación del cliente, pero debido a que esto pudiera cambiar en el futuro, se contempla que esto no afecte el diseño de los algoritmos.
  En el método **analize** se recoge esta información y conforme a esto se crean las estructuras de datos correspondientes, primeramente se diseña un contexto de temas (**Theme_Context**), el cual ordena eficientemente los temas siendo estos a la vez subtemas de otros temas, esto se diseña jerárquicamente.
  A la vez, los estudiantes se recopilan en **Student_Preferences**, donde se guardan los datos de los estudiantes en caso preciso y las respuestas de los mismos a las encuestas.
  Se analizan los estudiantes en pares no repetidos, se tiene la certeza que si se compara el estudiante i con el estudiante j se obtienen los mismos resultados que si se analiza al estudiante j con el i. Se comparan los criterios de preferencia de los estudiantes, qué tan alejados o cercanos están los mismos en sus gustos. Antes de explicar el proceso de comparación se pasa a explicar la organización de las preferencias de cada estudiante.
  Una vez que se recojan las respuestas de cada estudiante como (*Student_Preferences*) en **answers** (diccionario con llave *tema* y valor correspondiente: *respuesta del estudiante en ese tema*), se pasa a analizar sus preferencias (**preferences**) en orden descendente.
  Se implementa una clase **AVLSearchBinaryTree_Insert** la cual crea un AVL con valores en los nodos (*tema*, *cantidad de respuestas al tema*), mientras mayor cantidad de aciertos tenga ese tema (que se considera una hoja de la jerarquía de los temas, ya que los temas *padres* se van desglosando en subtemas *hijos*, los cuales a su vez tienen su propia jerarquía, hasta caer en un subtema *hoja*, en la cual los

estudiantes desarrollan o *marcan* sus gustos correspondientes), mayor es su preferencia. Estos nodos se organizan por el segundo valor de la tupla en forma ascendente. Luego de estar confeccionado el AVL, se realiza un recorrido entre-orden y devuelve una lista ordenada de las preferencias y las mismas se guardan en orden inverso en **preferences**.

Como por cada estudiante se tiene un orden en sus preferencias, se analizan los pares de estudiantes, esto se implementa en **invert_count**. Se toman **s1.prefences** y **s2.preferences** (**s1** y **s2** estudiantes) y se comparan, donde sin pérdida de generalidad se considera como arreglo ordenado a **s1.preferences**. Entonces, se ordena **s2.preferences** respecto a **s1.preferences**, y dependiendo de la cantidad de inversiones que tenga $tema_{i2}$ (el tema i de **s2**) con respecto a $tema_{i1}$ (tema i de **s1**) se establece la lejanía o cercanía del $tema_i$ que presentan ambos estudiantes. El criterio de cercanía (**s.likes**) o lejanía (**s.dislikes**), lo brinda un porcentaje, si la cantidad de inversiones es menor que un p porciento (no se define porque cada tema o encuesta tiene su propio patrón definido) entonces el $tema_i$ es común entre ambos estudiantes en su orden de preferencias. En caso contrario, los gustos de ambos estudiantes con respecto al tema están alejados. Después de conocer la relación entre ambos estudiantes se añade a las estructuras de datos correspondientes en la instancia de **s1** y **s2**.

Luego de recorrer y analizar los gustos de todos los pares de estudiantes, se recurre al análisis general estadístico, el cual se guarda en la instancia de **Theme_Context**, y se implementa en el método **theme_context.stadistics_result**.

- **hierarchy.py** : En dicho fichero se implementa la clase **Theme_Context** y **Theme_Scope**. En **Theme_Scope** se encuentra la jerarquía de los temas, y en **Theme_Context** se guardan estos temas y se realizan los análisis referentes a los mismos, como por ejemplo el análisis estadístico con respecto a las respuestas de los estudiantes.
- **likes_dislikes.py**: Se encuentra la clase **Student_Preferences**, donde se guardan los datos de cada estudiante, sus respuestas a las encuestas, la relación que guarda este con los *n - 1* estudiantes restantes, sus preferencias con respecto a los temas.
- **invert_array.py**: Se realiza el análisis **invert_count** mencionado anteriormente.
- **stadistics.py**: Se implementa la clase **Stadistics_Op**, la cual se encarga de realizar el análisis estadístico por tema. Se implementa el cálculo de las medidas de Tendencia Central y las medidas de Disper-

sión, necesarias para una correcta interpretación. Dicho análisis por tema se instancia en 2.1, donde se guarda en el diccionario **stadistics** (*tema* : *análisis estadístico*).

2.2 API Para implementar un sistema capaz de procesar y mostrar los resultados de una manera atractiva para el usuario final, se desarrolló una aplicación web que consta de dos partes, una aplicación visual implementada en React y otra aplicación de tipo cliente-servidor conocida como interfaz de programación de aplicaciones(API) desarrollada en Django.

Se diseñó un modelo de cuestionario con preguntas y respuestas, con temas y subtemas asociados a las preguntas, para las encuestas de tipo tiempo libre, además de modelos para los tests de creatividad y humor, y otro para gestionar el nivel de agrado de las asignaturas. Para esto se creó una app llamada **questionnaire** dentro de la API, que contiene los modelos:

- **Questionnaire** con dos propiedades, una *name* con el nombre del cuestionario y otr *pub_date* con la fecha de creación del cuestionario.
- **Themes** con un campo *name* con el nombre del tema, y otro *father* con el nombre del tema padre asociado a este mismo tema, de tener padre se convierte en subtema, de lo contrario se considera tema simplemente.
- **Question** tiene un campo *text* con la pregunta en cuestión, y dos llaves foráneas con el cuestionario y el tema o subtema al que pertenece, además de dos propiedades para ver la lista de opciones y respuestas asociadas a la pregunta.
- **Answer** con un campo *answer* con la respuesta de la pregunta, y dos llaves foráneas, una a la pregunta y otra al usuario que esta respondiendo, este modelo usuario lo discutiremos más adelante al igual de cómo se gestiona el anonimato de los encuestados.
- **Option** con un *value* con el valor de la opción y una llave foránea a la pregunta de la cual es opción.

Para los tests de creatividad y humor, y para gestionar la información de las asignaturas se creó una app llamada **averagetest** con los siguientes modelos:

- **AverageTest** este modelo contiene dos campos, uno llamado *category* para distinguir entre un test de humor y uno de creatividad, y otro *value* para guardar el resultado del test, y una llave foránea asociada al usuario que realizó el test.

- **SubjectTest** contiene un campo *name* con el nombre de la asignatura y otros tres campos *likes*, *dislikes* y *dontcare* con la cantidad de me gusta, no me gusta y me da igual de dicha asignatura.

Tenemos además los siguientes endpoints de los que consume la aplicación visual para que el usuario interactúe con el sistema y conozca los datos que le interesan de una manera atractiva:

√ **http://127.0.0.1:8000/questionnaire-create/** para crear un cuestionario de tipo tiempo libre.

√ **http://127.0.0.1:8000/questionnaire-response/** para guardar una respuesta de un cuestionario de tipo tiempo libre.

√ **http://127.0.0.1:8000/questionnaire-process/** para procesar los resultados de las encuestas de tipo tiempo libre, donde se recopilan los resultados de las encuestas realizadas y el conjunto de temas y subtemas registrados en el sistema, dicha información se le facilita al backend para su posterior análisis, estos resultados son guardados en files que se generan dentro de la API en un espacio destinado para ello.

√ **http://127.0.0.1:8000/process-most-popular-themes/** para procesar de acuerdo a la cantidad de votos los temas más populares, es donde se hace un análisis a partir de todas las encuestas realizadas y por cada selección de un subtema específico se contabiliza la cantidad de selecciones del tema padre que contiene dicho subtema, luego estos resultados se ordenan de manera descendente, quedando en el top los más votados, por ende los más populares y son devueltos a la aplicación visual en formato JSON.

√ **http://127.0.0.1:8000/process-most-popular-subthemes/** para procesar de acuerdo a la cantidad de votos los subtemas más populares, es donde se hace un análisis similar al explicado anteriormente, solo que en este caso se contabilizan los subtemas de manera independiente, luego estos resultados por subtemas se ordenan de manera descendente, quedando en el top los más votados, por ende los más populares y son devueltos a la aplicación visual en formato JSON.

√ **http://127.0.0.1:8000/process-average-test/creativity/** o **http://127.0.0.1:8000/process-average-test/humor/** para guardar una respuesta del test de creatividad o de humor según el tipo especificado en la url.

√ **http://127.0.0.1:8000/process-years-average/humor/** o **http://127.0.0.1:8000/process-years-average/creativity/** para procesar los resultados de los test de creatividad o de humor según

el tipo especificado en la url, aquí se recopilan todos los resultados del test en cuestión y se retorna en formato JSON el promedio de los datos.

√ **http://127.0.0.1:8000/process-years-particularities/humor/** o **http://127.0.0.1:8000/process-years-particularities/creativity/** para procesar los resultados de los test de creatividad o de humor según el tipo especificado en la url, es donde se recopilan todos los resultados del test en cuestión y se contabilizan y clasifican en las distintas categorías que serán descritas posteriormente, luego son devueltos a la aplicación en formato JSON las puntuaciones obtenidas en cada categoría.

√ **http://127.0.0.1:8000/process-subject-test/** para guardar una respuesta del test de asignaturas, es donde el usuario decide se le gustan, le disgustan o le da igual cada una de las asignaturas del test.

√ **http://127.0.0.1:8000/process-subject-test-result/Algebra/** para procesar los resultados del test de asignaturas para una asignatura específica, aquí se recopilan todos los resultados obtenidos en esa asignatura y se retorna a la aplicación visual en formato JSON la cantidad de likes, dislikes y dontcares.

√ **http://127.0.0.1:8000/process-subject-test-all-result/** para procesar los resultados del test de asignaturas de todas las asignaturas, es donde se recopilan todos los resultados obtenidos en cada una de las asignaturas y se retorna en formato JSON la cantidad de "likes", "dislikesz "dontcares"de todas estas.

Por otro lado se trabajó en una aplicación llamada **user** con el objetivo de crear un proyecto más robusto y extensible para su posterior desarrollo. Se creó un modelo usuario con un conjunto de campos como *name*, *last_name*, *email* y otras propiedades para controlar los permisos del usuario, códigos de verificación para temas de autenticación, además de una tabla **role** para gestionar los roles de los usuarios y permitir que la aplicación cuente con roles como estudiante, profesor y administrador. Se implementaron, además, endpoints para el log-in y registro de usuarios a la aplicación, cambio y recuperación de contraseña, y para la autorización se trabajó con JWT.

Se trataron como anónimas las encuestas por tanto todos los users creados en la aplicación son falsos, se utilizan solamente para controlar el número de encuestados y distinguir entre las distintas respuestas. No obstante, como se menciona anteriormente, las bases estan creadas para el caso en el que se quiera establecer un sistema de registro para los estudiantes, dintinción de roles y permisos. Para el acceso a la API se

trabajó con *superusers* que se crearon a nivel de consola y se registraron en la aplicación, capaces de navegar por la API y gestionarla a su totalidad; pues el framework incluye vistas genéricas para trabajar con los modelos, lo que permite crear, editar, borrar y trabajar de manera general con los modelos antes mencionados.

2.3 Front-End Con la intención de crear vistas interactivas y de fácil manejo tanto para los usuarios como para los desarrolladores, se utiliza la biblioteca ReactJS. Para la creación de los cuestionarios que conforman las encuestas, se hace uso de la biblioteca SurveyJS que provee herramientas para previsualizar los test, así como para obtener los códigos asociados a la creación de los componentes del mismo, así como para las respuestas que se envían al completarlos. La variedad de componentes que incluye esta biblioteca permite modificar los formatos de las preguntas en caso de que el cliente lo requiera.
En aras de simplificar el proceso de maquetación, para el embellecimiento de las vistas se utilizó el framework Bootstrap, ya que permite utilizar muchos elementos web desde combinando HTML5, CSS y Javascript.
El proceso de enrutamiento se gestiona haciendo uso de React-Router-DOM, ya que aporta agilidad y ligereza a la aplicación con el enrutamiento dinámico gracias a los componentes, renderizando hasta un componente en dependencia de la ruta. Para el manejo de request y response se hace uso de la biblioteca Axios.
En el caso de los resultados de los test de Creatividad y Humor, se utiliza Chart.js para la creación de los gráficos.
Para todas las elecciones se tuvo en cuenta que exitiera una comunidad de desarrolladores y documentación extensa, de manera que resulte cómodo hacer investigaciones sobre las herramientas, sus ventajas y características.

2.4 Cómo poner a funcionar la aplicación La aplicación en su totalidad se compone de dos partes: una en la que se ejecutará el servidor y otra donde se ejecutará el cliente. En nuestro caso, el cliente será la aplicación visual, que consumirá de los recursos del Back-End a través de la comunicación que se establece con la API.

FRONT-END

En la carpeta "App" se encuentra nuestra solución a la aplicación visual y, dentro de esta, se deberá, primeramente, instalar las dependencias.

```
npm install
```

Para "levantar" los servicios de la aplicación visual será necesario ejecutar el siguiente comando:

```
npm start
```

Los resultados se mostrarán en **http://localhost:3000/**.

BACK-END:

Requisitos:

- Python: `https://docs.python.org/3/`
- Postgresql: `https://www.postgresql.org/docs/12/tutorial-install.html`
- Django Rest Framework: `https://docs.djangoproject.com/en/4.0/topics/install/`

Ejecución:

Es necesario, inicialmente, migrar para generar las estructuras de la base de datos correspondiente, para esto deberá ejecutarse:

```
python manage.py makemigrations
pyhton manage.py migrate
```

Una vez creada la base de datos ya se puede ejecutar la API:

```
python manage.py runserver
```

Si se desea consumir de la API desde un IP diferente se recomienda *ngrok*, para más información consultar[1]. Además se puede utilizar Postman[2] para probar todos los endpoints que contiene la API y fueron descritos anteriormente, para más facilidad el trabajo trae adjunto la colección utilizada por nosotros para el proceso de prueba de la aplicación.

Referencias

1. https://ngrok.com/docs
2. https://www.postman.com/