


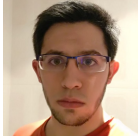



Almacenamiento de votos

Grupo 2

ID de Opera: 21

Miembros del grupo:

Nombre	Implicación	Foto
Camero Ruiz, Elena	5	
Carmona Oliva, Marta	5	
Martínez Quiñones, José Luis	5	
Serrano Ramos, Pedro	5	
Vázquez Argumedo, Jesús	5	

Enlaces de interés:

- [Repositorio interno](#)
- [Repositorio integrado](#)
- [Portal de Opera](#)
- [Wiki del grupo](#)

Índice

<u>Resumen</u>	3
<u>Introducción y contexto</u>	4
<u>Descripción del sistema</u>	5
Modelo de datos	5
API	5
<u>Planificación del proyecto</u>	13
Milestone 1: Ecosistemas preparados	13
Milestone 2: Sistema funcionando con incremento	14
Milestone 3: Taller de automatización	17
Milestone 4: Entrega y defensa de trabajos	20
<u>Entorno de desarrollo</u>	23
<u>Gestión del cambio, incidencias y depuración</u>	25
Incidencias Internas	25
Incidencias Externas	26
<u>Gestión del código fuente</u>	29
Gestión de commits	29
Usage Model	30
<u>Gestión de la construcción e integración continua</u>	31
<u>Gestión de liberaciones, despliegue y entregas</u>	32
Política de nombrado e identificación de los entregables	32
Proceso definido para las entregas	32
Proceso definido para liberaciones y despliegue	32
<u>Mapa de herramientas</u>	33
<u>Ejercicio de propuesta de cambio</u>	34
<u>Conclusiones y trabajo futuro</u>	35
<u>Anexo: Informe de Toggl</u>	36

Resumen

Nuestro trabajo consiste en implementar el módulo referente al almacenamiento de votos del proyecto nVotes, que debe integrarse posteriormente con los demás subsistemas para crear un portal de votación electrónica.

En un sistema de votación clásica, nuestro módulo sería el equivalente a las urnas que contienen los votos depositados por los votantes. Dado que la votación debe ser secreta, se debe garantizar la confidencialidad y la integridad de los votos. Al mismo tiempo también debemos ser capaces de relacionar cada voto con la persona que lo realizó, así como las preguntas que le corresponden y las respuestas que dio. Todo esto debe hacerse sin perder en ningún momento la confidencialidad de dichos votos.

Nuestra propuesta de solución ha sido, en primer lugar, crear una base de datos con una tabla principal llamada “votos”, la cuál contiene los siguientes atributos: identificador del objeto voto, token del usuario, token de la votación, token de la pregunta, y token de la respuesta. Para todos los atributos referentes al voto se utilizan tokens cifrados para garantizar la anonimización necesaria mencionada anteriormente. De esta manera tenemos una relación entre el usuario que realizó el voto, la respuesta que dio y la pregunta a la que correspondía sin perder la confidencialidad.

En segundo lugar, se implementó una API para gestionar el acceso a nuestra base de datos y la comunicación con las bases de datos de otros subsistemas. Aunque los tokens de respuesta recibidos por nuestra base de datos vienen ya previamente cifrados por los subsistemas correspondientes, se decidió realizar una segunda capa de encriptación a esos tokens de respuesta, para proporcionar así una mayor seguridad al sistema general.

Una vez terminadas las partes fundamentales de nuestro subsistema, se procedió a la realización de pruebas unitarias y funcionales para comprobar el correcto funcionamiento de nuestra base de datos y nuestra API y detectar los posibles errores, de manera que pudiéramos corregirlos antes de integrar nuestro módulo con los demás.

Posteriormente, se implementó el código necesario para la automatización de la instalación de todas las herramientas y dependencias de nuestro subsistema, para simplificar así la unificación con el resto de subsistemas.

Finalmente procedimos a la elaboración de la documentación final referente a nuestro módulo y al despliegue del proyecto en el sistema general.

Introducción y contexto

El objetivo principal de nuestro subsistema, como se mencionó anteriormente, es obtener todos los votos de una votación y almacenarlos de manera segura y anónima pero ofreciendo también la posibilidad de relacionar a cada votante con sus votos realizados, ya que esa información debe ser accesible para otros módulos que forman parte del sistema.

Nuestro módulo debe comunicarse principalmente con el subsistema de autenticación, el subsistema de administración votaciones, los subsistemas de cabinas de votación, y el subsistema de recuento. En cuanto al subsistema de autenticación, necesitamos obtener los datos de ese módulo para que un usuario pueda verificar si su voto se registró correctamente en el sistema. Por otro lado, el subsistema de administración de votaciones se encarga de gestionar la creación/edición/borrado de las votaciones y por tanto son los que establecen todos los atributos que se guardarán de una votación, aunque nuestra API no se comunica directamente con la suya, en la fase inicial nos hizo falta saber como sería la estructura de su base de datos para decidir que datos eran relevantes para nosotros y poder mantener una cierta uniformidad entre los subsistemas. Por otro lado, nuestra API se comunica con las API de las cabinas de votaciones para obtener todos los votos existentes en el sistema y almacenarlos en nuestra base de datos, de manera que cada usuario tenga asociadas las respuestas que dio a cada pregunta de cada votación pero sin perder la confidencialidad. Finalmente, nuestro subsistema también debe comunicarse con el de recuento, ya que nuestra API les ofrece los votos con la información necesaria para realizar el recuento.

Por tanto, en la realización de nuestro modelo de datos y de nuestra API se ha tenido en cuenta en todo momento las relaciones con los demás subsistemas para garantizar así la correcta integración del sistema completo.

Descripción del sistema

Modelo de datos

Los datos que se van a almacenar en nuestra base de datos son:

- Tabla votos:

```
- id: (int) identificador del voto [pk, autoincrementable]
- token_usuario: (String) identificador del usuario que realiza el voto [fk, not null]
- token_pregunta: (String) identificador de la pregunta a la que está respondiendo [fk, not null]
- token_respuesta: (String) identificador de la respuesta [fk]
- token_votacion: (String) identificador de la votacion [fk]
```

- Los datos relativos a la votación los recibiremos cifrados, y para garantizar la anonimización de cada voto añadiremos una segunda capa de encriptación al token de las respuestas antes de almacenarlos.

- Tabla tokens:

```
- id: (int) identificador del token [pk, autoincrementable]
- token_bd: (String) Token para autenticarse y acceder a nuestra BBDD
```

API

Comprobar voto

URL:

```
https://almacenamiento.nvotesus.es/api/get/comprobar_voto_pregunta/{token_bd}/{token_usuario}/{token_votación}/{token_pregunta}
```

Descripción:

Se comprueba si un determinado usuario ha votado o no en esa votación.

Dependencias:

Autenticación lo puede usar para que un usuario pueda comprobar, una vez logueado, si su voto se ha realizado correctamente.

Petición HTTP:

GET

Parámetros de petición:

- * token_bd: Token para verificar que el usuario que realiza la petición tiene permiso de acceso a nuestra base de datos (String)
- * token_usuario: Identificador del usuario (String)
- * token_votación: Identificador de la votación (String)

Ejemplo de petición:

https://almacenamiento.nvotesus.es/api/get/comprobar_voto/1/1/1

Respuesta:

Nombre	Tipo	Descripción
result	JSON	Objeto voto
msg	String	200 OK
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	401 NOT FOUND El usuario no ha realizado ningun voto en esta votacion.
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	401 UNAUTHORISED Token incorrecto.

Ejemplo de respuesta:

```
{
  "1": {
    "id": "...",
    "token_usuario": "1",
    "token_pregunta": "...",
    "token_respuesta": "..."
  }
}
{
  "code": "404 NOT FOUND"
  "msg": "El usuario no ha realizado ningun voto en esta votacion."
}
{
  "code": "404 UNAUTHORISED"
  "msg": "Token incorrecto."
}
```

Comprobar voto pregunta

URL:

https://almacenamiento.nvotesus.es/api/get/comprobar_voto_pregunta/{token_bd}/{token_usuario}/{token_votación}/{token_pregunta}

Descripción:

Se comprueba si un determinado usuario ha votado o no en una determinada pregunta de una votación.

Dependencias:

Autenticación lo puede usar para que un usuario pueda comprobar, una vez logueado, si su voto se ha realizado correctamente.

Petición HTTP:

GET

Parámetros de petición:

```
* token_bd: Token para verificar que el usuario que realiza la
petición tiene permiso de acceso a nuestra base de datos (String)
* token_usuario: Identificador del usuario (String)
* token_votación: Identificador de la votación (String)
* token_pregunta: Identificador de la pregunta (String)
```

Ejemplo de petición:

https://almacenamiento.nvotesus.es/api/get/comprobar_voto_pregunta/1/1/1/1

Respuesta:

Nombre	Tipo	Descripción
result	JSON	Objeto voto
msg	String	200 OK
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	404 NOT FOUND El usuario no ha realizado ningun voto en esta pregunta de esta votacion.
Nombre	Tipo	Descripción
result	Boolean	Código de error

msg	String	401 UNAUTHORISED Token incorrecto.
-----	--------	------------------------------------

Ejemplo de respuesta:

```
{
  "1": {
    "id": "...",
    "token_usuario": "1",
    "token_pregunta": "1",
    "token_respuesta": "..."
  }
}
{
  "code": "401 NOT FOUND"
  "msg": "El usuario no ha realizado ningun voto en esta votacion."
}
{
  "code": "404 UNAUTHORISED"
  "msg": "Token incorrecto."
}
```

Almacenar voto

URL:

https://almacenamiento.nvotesus.es/api/post/almacenar_voto

Parámetros de petición:

- * token_bd: Token para verificar que el usuario que realiza la petición tiene permiso de acceso a nuestra base de datos (String)
- * token_usuario: Identificador del usuario (String)
- * token_votacion: Identificador de la votación (String)
- * token_pregunta: Identificador de la pregunta (String)
- * token_respuesta: Identificador de la respuesta (String)

Descripción:

Almacena el voto pedido en la base de datos.

Dependencias:

Debemos recibir estos datos de las cabinas de votaciones.

Petición HTTP:

POST

Ejemplo de petición:

https://almacenamiento.nvotesus.es/api/post/almacenar_voto/

Respuesta:

Nombre	Tipo	Descripción
result	Boolean	Código de éxito
msg	String	200 OK El voto se ha almacenado satisfactoriamente.
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	400 BAD REQUEST Un usuario sólo puede votar una vez a una pregunta.
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	401 UNAUTHORISED Token incorrecto.

Ejemplo de respuesta:

```
{
  "code": "200 OK"
  "message": "El voto se ha almacenado satisfactoriamente."
}
{
  "code": "400 BAD REQUEST"
  "message": "Un usuario sólo puede votar una vez a una
pregunta."
}
{
  "code": "404 UNAUTHORISED"
  "msg": "Token incorrecto."
}
```

Almacenar voto múltiple

URL:

https://almacenamiento.nvotesus.es/api/post/almacenar_voto_multiple

Parámetros de petición:

- * token_bd: Token para verificar que el usuario que realiza la petición tiene permiso de acceso a nuestra base de datos (String)
- * token_usuario: Identificador del usuario (String)
- * token_votacion: Identificador de la votación (String)

* token_voto: Identificador del voto, es un array que contiene el token de la pregunta y el token de la respuesta (String)

Descripción:

Almacena el voto pedido en la base de datos.

Dependencias:

Debemos recibir estos datos de las cabinas de votaciones.

Petición HTTP:

POST

Ejemplo de petición:

https://almacenamiento.nvotesus.es/api/post/almacenar_voto_multiple/

Respuesta:

Nombre	Tipo	Descripción
result	Boolean	Código de éxito
msg	String	200 OK El voto se ha almacenado satisfactoriamente.
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	400 BAD REQUEST Un usuario sólo puede votar una vez a una pregunta.
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	401 UNAUTHORISED Token incorrecto.

Ejemplo de respuesta:

```
{
  "code": "200 OK"
  "message": "El voto se ha almacenado satisfactoriamente."
}
{
  "code": "400 BAD REQUEST"
  "message": "Un usuario sólo puede votar una vez a una
pregunta."
}
{
  "code": "404 UNAUTHORISED"
  "msg": "Token incorrecto."
}
```

```
}
```

Obtener votos

URL:

https://almacenamiento.nvotesus.es/api/get/obtener_todo/{token_bd}/{token_votacion}/{token_pregunta}

Parámetros de petición:

- * token_bd: Token para verificar que el usuario que realiza la petición tiene permiso de acceso a nuestra base de datos (String)
- * token_votación: Identificador de la votación (String)
- * token_pregunta: Identificador de la pregunta (String)

Descripción:

Obtener todos los votos de una pregunta en una determinada votación.

Dependencias:

Este método lo usa Recuento para poder obtener todos los votos.

Ejemplo de petición:

https://almacenamiento.nvotesus.es/api/get/obtener_todo/1/1/1

Respuesta:

Nombre	Tipo	Descripción
result	JSON	Objetos votos
msg	String	200 OK
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	404 NOT FOUND
Nombre	Tipo	Descripción
result	Boolean	Código de error
msg	String	401 UNAUTHORISED Token incorrecto.

Ejemplo de respuesta:

```
{  
  "1": {  
    "id": "...",
```

```
        "token_usuario": "...",
        "token_pregunta": "...",
        "token_respuesta": "..."
    }
    "2": {
        "id": "...",
        "token_usuario": "...",
        "token_pregunta": "...",
        "token_respuesta": "..."
    }
}
{
    "code": "404 NOT FOUND"
    "message": "..."
}
{
    "code": "404 UNAUTHORISED"
    "msg": "Token incorrecto."
}
```

Planificación del proyecto

Milestone 1: Ecosistemas preparados

Para la correcta realización del proyecto, lo primero que hicimos fue establecer los roles de los distintos miembros del grupo y el reparto de tareas que realizaríamos en la primera iteración. También se acordó una plantilla de documentación para la creación de actas de reunión que recogieran la información relevante que discutiéramos en las reuniones y que utilizaríamos en posteriores iteraciones. Posteriormente procedimos a la elección de las herramientas y del entorno de desarrollo que usaríamos para implementar nuestro subsistema, así como las herramientas que utilizaríamos para la gestión de tareas y tiempo. Una vez instaladas las herramientas y montado el entorno, se elaboró la wiki referente a nuestro subsistema con la información que teníamos hasta el momento y las decisiones que habíamos tomado, para facilitar así la comunicación con los demás subsistemas.

Para la gestión de la comunicación se utiliza un grupo de Telegram, en el cuál usamos un bot de GitHub que nos informa de todos los commits y notificaciones relevantes de nuestro repositorio interno y de nuestro repositorio integrado. Para la gestión de tareas y las incidencias internas utilizamos [Trello](#). Para la gestión del tiempo se utilizó Toggl, donde se mide el tiempo que cada miembro del grupo dedica a las distintas tareas del proyecto.

La página principal de nuestro portal de Trello contiene 5 tableros cuyas funcionalidades son las siguientes:

- **Tareas pendientes:** En este tablero se añadirán las tareas que vayan surgiendo y que no estén asignadas ni realizadas aún.
- **Tareas en proceso:** En este tablero estarán las tareas asignadas que están en proceso de ejecución.
- **Tareas en revisión:** En este tablero estarán las tareas acabadas para que sean revisadas antes de darlas por finalizadas.
- **Tareas realizadas:** En este tablero estarán las tareas ya completadas.
- **Información:** Este tablero no pertenece al proceso de realización de tareas, simplemente contiene información relevante al proyecto:
 - *Método de revisión:* Se decidió que cada miembro del grupo se encargaría de revisar la correcta realización de las tareas por parte de los demás miembros del grupo. Se realiza de forma circular.
 - *Próxima reunión:* En esta tarjeta se informa sobre la fecha de la próxima reunión así como las fechas de las defensas cuando es necesario.
 - *Instalación del software:* Información relativa a las herramientas que utilizamos y como instalarlas.
 - *Realización de pruebas:* Aquí se establece el reparto a la hora de realizar las pruebas y el formato que utilizaríamos para que todo quede uniforme.

Al comienzo de cada iteración se realiza una reunión de planificación en la que se decide el reparto de tareas y posteriormente el coordinador del grupo asigna a cada miembro las tareas correspondiente. En las tarjetas pueden haber checklists en caso de existir subtareas, así como enlaces a la información necesaria para la realización de dicha tarea.

Milestone 2: Sistema funcionando con incremento

Reunión de planificación del Milestone 2 (23/11/17):

1º punto del día: Se ha revisado cada apartado de la wiki para comprobar toda la información que hay escrita y saber que hay que modificar o añadir.

- **Conclusión:** Es necesario seguir añadiendo más información a la wiki a medida que se va obteniendo más información.

2º punto del día: Se ha detallado la información que hay que añadir en cada uno de los apartados que faltan o en los que es necesario añadir algo más de nuestra wiki. •
Conclusión: Hay que rellenar dicha información en la wiki.

- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 24/11/2017.

3º punto del día: Se ha creado un formato de incidencias.

- **Conclusión:** Hay que detallar mejor el formato.

4º punto del día: Se han especificado las tecnologías que vamos a usar, ya que en la anterior reunión no se llegó a especificar todas las necesarias, siendo éstas las siguientes: Python como lenguaje de programación, PyCharm como entorno de desarrollo, MySQL como base de datos y SQLite como biblioteca necesaria para el desarrollo.

5º punto del día: Se ha analizado los datos que vamos a poder necesitar y se ha decidido usar como modelo de datos un id de usuario, un id de la pregunta y un id del voto.

6º punto del día: Se ha analizado las distintas posibles llamadas a la API que se puede realizar y hemos llegado a la conclusión de que todas las llamadas a la API la harán a nuestra base de datos para obtener mayor seguridad en el sistema de votación. Se ha determinado un formato sin muchos detalles por el momento y nuestras tres llamadas a la API, que son: comprobar voto, pedir voto y obtener voto.

- **Conclusión:** Se debe de especificar con más detalles la estructura de la API.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 28/11/2017.

7º punto del día: Se ha decidido que Pedro cree un repositorio de Github en el sitio creado para los grupos creado por integración, en el cual se incluirá todo el código que esté funcionando para cualquier Milestone.

- **Conclusión:** Se debe de crear el repositorio Github.
- **Planes de acción:** Se le asigna la tarea a Pedro.
- **Plazo:** El día 30/11/2017.

8º punto del día: Se ha planificado y asignado las tareas a realizar para cada miembro del grupo de manera más equitativamente posible.

Reunión de planificación del Milestone 2 (26/11/17):

1º punto del día: Jesús se ha reunido con los demás coordinadores del grupo y ha comunicado todos los detalles que se ha hablado en la reunión, siendo los temas principales una estructura más o menos clara de la API, aclaración de nuestras llamadas a la API y estado que debe de tener nuestro proyecto.

- **Conclusión:** Ahora que se han aclarado muchas dudas, podemos empezar a generar código.

2º punto del día: Se ha comprobado el estado de la wiki y comentado que es necesario cambiar y/o añadir.

- **Conclusión:** Se ha detallado más nuestro modelo de datos.

3º punto del día: Se ha asignado a Jesús la creación de una base de datos para nuestro proyecto.

- **Conclusión:** Hay que crear una base de datos.
- **Planes de acción:** Se le asigna la tarea a Jesús.
- **Plazo:** El día 30/11/2017.

4º punto del día: Se ha comentado como los detalles de nuestra API para la hora de realizar la documentación y creación de la misma.

- **Conclusión:** Se debe de detallar la estructura de la API en la wiki tal y como se ha acordado en la reunión.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 28/11/2017.

5º punto del día: Como ya se ha especificado con más detalles el funcionamiento de nuestra API, una vez desarrollada la documentación de la misma, se podrá dar comienzo a la generación del código.

- **Conclusión:** Se debe de crear los métodos de llamada a la API acorde con la documentación genera en la wiki.

- **Planes de acción:** Se le asigna la tarea a Marta.

- **Plazo:** El día 30/11/2017.

6º punto del día: Se ha debatido un poco el funcionamiento de Docker y se ha acordado que se debe investigar un poco más a la hora de emplear dicha herramienta.

- **Conclusión:** Se debe de estudiar el correcto funcionamiento de Docker.

- **Planes de acción:** Se le asigna la tarea a José Luis.

- **Plazo:** El día 3/12/2017.

7º punto del día: Se ha propuesto un método de revisión del trabajo para que todos nos obliguemos a revisar una parte mínima del trabajo para que haya el mínimo número de errores en el proyecto, quedando de la siguiente manera: Elena revisa a Jesús, Jesús revisa a Pedro, Pedro revisa a Marta, Marta revisa a José Luis y José Luis revisa a Elena.

- **Conclusión:** Se ha establecido un método de revisión.

Reunión de revisión del Milestone 2 (30/11/17):

1º punto del día: Se ha comprobado el estado del proyecto y hemos visto que aunque no haya ningún código subido al repositorio, está todo casi terminado a falta de algunas dudas que se resuelven en los siguientes puntos del día.

- **Conclusión:** Al proyecto le falta poco para completar nuestros objetivos del Milestone 2.

2º punto del día: Se ha comprobado el estado de la wiki y hemos definido un formato más detallado para documentar la API de manera que quede todo mucho más claro.

- **Conclusión:** Se debe detallar en la wiki el formato de la API tal y como se ha acordado.

- **Planes de acción:** Se le asigna la tarea a Elena.

- **Plazo:** El día 03/12/2017

3º punto del día: Se ha comprobado el funcionamiento de la base de datos y funciona correctamente. Tan solo habría que subir el código al repositorio. Además se ha propuesto incluir para el siguiente Milestone un token de seguridad de manera que deba introducirlo en la URL a la hora de hacer la llamada a la API, de modo que sólo los propietarios de dicho token puedan realizar la llamada.

- **Conclusión:** Se debe de crear una base de datos para los nuevos tokens y crear un método de comprobación.

- **Planes de acción:** Se le asigna la tarea a Jesús.
- **Plazo:** El día 11/12/2017.

4º punto del día: Se ha comprobado el funcionamiento de la API (tenía una copia de la base de datos) y funciona todo correctamente menos un par de errores forzados no contemplados. Se ha corregido en el momento y tan solo habría que subir el código al repositorio. Además como se ha propuesto la inclusión de los tokens, se debe incluir la propuesta para el siguiente Milestone.

- **Conclusión:** Se debe de incluir en las llamadas a la API los nuevos tokens.
- **Planes de acción:** Se le asigna la tarea a Marta.
- **Plazo:** El día 17/12/2017.

5º punto del día: Entre todos hemos rellenado el formulario que se debe entregar al profesor del Milestone 2.

- **Conclusión:** Se debe de entregar al profesor.
- **Plazo:** El día 04/12/2017.

Milestone 3: Taller de automatización

Reunión de planificación del Milestone 3 (12/12/17):

1º punto del día: Se ha comprobado el estado de la wiki y está todo correcto a falta de incluir la información de los tokens de la base de datos en la API.

- **Conclusión:** Se debe añadir la información respecto a los tokens en la wiki además de la información referida en los puntos 2, 3 y 4.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 16/12/2017

2º punto del día: Se ha comprobado el funcionamiento de la base de datos y se ha determinado cambiar el formato de dato de manera que todos los datos sean String.

- **Conclusión:** Se debe de cambiar el formato de los datos de la base de datos.
- **Planes de acción:** Se le asigna la tarea a Jesús.
- **Plazo:** El día 16/12/2017.

3º punto del día: Se ha comprobado el funcionamiento de la API y se decide que también hay que cambiar obviamente el formato en el que se recibe los datos de la API.

- **Conclusión:** Se debe de cambiar el formato de los datos de entrada de la API.
- **Planes de acción:** Se le asigna la tarea a Marta.
- **Plazo:** El día 20/12/2017.

4º punto del día: Se ha decidido que tenemos que crear varias ramas para poner en práctica la materia de la asignatura y para organizar mejor el código, añadiendo así una rama de base de datos y otra rama de API, dejando la rama master para unir ambas ramas.

- **Conclusión:** Se debe de añadir una rama DB y otra rama API.
- **Planes de acción:** Se le asigna la tarea a Jesús y a Marta a las respectivas ramas.
- **Plazo:** El día 14/12/2017.

5º punto del día: Se ha decidido que tenemos que crear un script en el que se instale todas las herramientas que necesitamos para nuestro proyecto ejecutando dicho script sin tener que hacer uso del tutorial creado anteriormente.

- **Conclusión:** Se debe de añadir un fichero de dependencia.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 20/12/2017.

6º punto del día: Se ha decidido vamos a volver a encriptar los datos recibidos en el método almacenar voto, teniendo así una capa más de encriptación y dándole más seguridad a los votos.

- **Conclusión:** Se debe de encriptar los datos recibido.
- **Planes de acción:** Se le asigna la tarea a Pedro.
- **Plazo:** El día 20/12/2017.

7º punto del día: Se ha decido que se va a incluir el proyecto en la imagen de Docker, puesto que ya se ha investigado bien cómo usarlo.

- **Conclusión:** Se debe de añadir una imagen de Docker al proyecto.
- **Planes de acción:** Se le asigna la tarea a José Luis.
- **Plazo:** El día 20/12/2017.

8º punto del día: Jesús recomienda probar la máquina virtual facilitada por el equipo de integración.

Reunión de revisión del Milestone 3 (20/12/17):

1º punto del día: Se ha comprobado el estado del proyecto y falta aún la creación de la imagen de docker y la encriptación de los datos en el que se ha decidido posponer estas dos tareas para el siguiente Milestone. Todo lo demás cumple con las expectativas esperadas para este Milestone.

- **Conclusión:** Falta la imagen de docker y la encriptación de datos.
- **Planes de acción:** Se le asigna la tarea a José Luis y a Pedro respectivamente.
- **Plazo:** Para el Milestone 4.

2º punto del día: Se ha estudiado los tipos de pruebas que podemos realizar y hemos llegado a la conclusión que vamos a realizar pruebas funcionales y unitarias.

- **Conclusión:** Se van a realizar pruebas funcionales y unitarias.

3º punto del día: Se ha llegado a un acuerdo en cómo hacer las pruebas poniendo un formato específico y creando una plantilla en un documento para reflejar todas las pruebas realizadas además del código.

- **Conclusión:** Se debe de crear una plantilla donde se reflejen las pruebas una vez realizadas.
- **Planes de acción:** Se le asigna la tarea a Marta y Elena.
- **Plazo:** El día 20/12/2017.

4º punto del día: Se ha decidido que pruebas se van a realizar y se ha repartido las pruebas que van a realizar cada uno, repartiendo dos pruebas a cada uno y tres a otro.

- **Conclusión:** Se ha repartido las pruebas que deben de realizar cada uno.
- **Planes de acción:** Se le asigna la tarea a Jesús, Marta, Elena y José Luis.
- **Plazo:** El día 21/12/2017.

5º punto del día: Se ha decidido que es necesario crear una rama de pruebas para tener separado el código de pruebas respecto a las otras ramas.

- **Conclusión:** Se debe de añadir una nueva rama.
- **Planes de acción:** Se le asigna la tarea a Jesús.
- **Plazo:** El día 20/12/2017.

6º punto del día: Se ha rellenado el documento del Milestone 3 entre todos los miembros del grupo para entregárselo al día siguiente al profesor.

- **Conclusión:** Se debe de entregar el documento rellenado del Milestone 3.

- **Planes de acción:** Se le asigna la tarea a Jesús.
- **Plazo:** El día 21/12/2017.

Milestone 4: Entrega y defensa de trabajos

Reunión de planificación del Milestone 3 (29/12/2017):

1º punto del día: Se ha comprobado el estado del proyecto y falta aún la creación de la imagen de docker y la encriptación de los datos.

- **Conclusión:** Falta la imagen de docker y la encriptación de datos.
- **Planes de acción:** Se le asigna la tarea a José Luis y a Pedro respectivamente.
- **Plazo:** El día 12/01/2018.

2º punto del día: Se ha decidido mejorar las pruebas de tal manera que queden más intuitivas y se visualicen mejor.

- **Conclusión:** Se debe mejorar las pruebas.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 08/01/2018.

3º punto del día: Pedro ha planteado una serie de preguntas sobre la encriptación de los datos y la hemos ido resolviendo de tal manera que se le han resuelto todas sus dudas para poder seguir realizando su trabajo.

Reunión de revisión del Milestone 3 (07/01/2018):

1º punto del día: Se ha comprobado el estado del proyecto y se han realizado pequeños avances respecto a la anterior reunión debido a las vacaciones.

2º punto del día: Se han leído los nuevos requisitos y es necesario realizar nuevos cambios en nuestra wiki para que esté acorde a la wiki del profesor.

- **Conclusión:** Se debe realizar cambios en la wiki.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 12/01/2018.

3º punto del día: Se ha comprobado el estado de la encriptación y hemos visto que faltan algunos detalles para la finalización del código.

- **Conclusión:** Se debe finalizar la encriptación de datos.
- **Planes de acción:** Se le asigna la tarea a Pedro.

- **Plazo:** El día 12/01/2018.

4º punto del día: Se ha comprobado el estado de la imagen de docker y hemos visto que solamente tiene una pequeña para poder finalizarlo.

- **Conclusión:** Se debe finalizar la creación de la imagen de docker.
- **Planes de acción:** Se le asigna la tarea a José Luis.
- **Plazo:** El día 12/01/2018.

5º punto del día: Se ha decidido mejorar las pruebas realizadas para que no se produzca ningún tipo de error al ejecutar el script de pruebas. Para ello se buscara un framework o una herramienta que lo facilite, de manera que quede más claro y eficiente.

- **Conclusión:** Se debe buscar e implementar un framework que facilite el uso de pruebas.
- **Planes de acción:** Se le asigna la tarea a Marta.
- **Plazo:** El día 12/01/2018.

Reunión de finalización del Milestone 4 (12/01/2018)

1º punto del día: Se ha comprobado el estado del proyecto y se ha comprobado que solo hace falta realizar los cambios que se propongan en esta reunión.

2º punto del día: Se han leído los nuevos requisitos con más detenimiento para resolver algunas dudas sobre algunos puntos y se ha decidido que vamos a realizar un documento para entregarlo con toda la información de la wiki más algunos datos adicionales para que sea aún más completo.

- **Conclusión:** Se debe realizar cambios en la wiki y crear un documento del proyecto.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** El día 14/01/2018.

3º punto del día: Se ha comprobado el estado de la encriptación y hemos visto que ya está listo y funcionando correctamente en el repositorio.

4º punto del día: Se ha comprobado el estado de la imagen de docker y hemos visto que ya sabe implementar docker y solamente queda crear la imagen de nuestro proyecto cuando finalice.

- **Conclusión:** Se debe de crear la imagen de docker.
- **Planes de acción:** Se le asigna la tarea a José Luis.

- **Plazo:** Cuando finalice nuestro proyecto.

5º punto del día: Se ha comprobado los diferentes métodos que ha encontrado Marta para el correcto empleo de las pruebas y hemos decidido usar uno de ellos aunque Marta va a mejorar el uso del framework.

- **Conclusión:** Se debe implementar el framework seleccionado para el uso de pruebas.
- **Planes de acción:** Se le asigna la tarea a Marta.
- **Plazo:** El día 14/01/2018.

5º punto del día: Se ha decidido que el despliegue del proyecto lo realice Elena en vez de Jesús, ya que éste no puede usar la máquina virtual de Linux.

- **Conclusión:** Se debe desplegar el proyecto en el repositorio de integración.
- **Planes de acción:** Se le asigna la tarea a Elena.
- **Plazo:** Cuando finalice el proyecto.

7º punto del día: Como Pedro ha terminado su trabajo ha decidido implementar Travis en nuestro proyecto ya que es uno de los requisitos de la asignatura.

Entorno de desarrollo

El entorno de desarrollo utilizado para el proyecto es una maquina virtual de Ubuntu (versión 16.04.3) donde se instaló el IDE PyCharm Community (versión 2017.3) ya que hemos usado Python (versión 2.7) como lenguaje principal de desarrollo. Como sistema de gestión de bibliotecas se utilizó PIP. Se elaboró un documento de requirements.txt donde se especifican las bibliotecas a instalar. También contamos con un script de bash, prestart.sh, el cuál ejecuta el comando para instalar todas las bibliotecas listadas en el documento de requerimientos. El servidor empleado para la base de datos es el MySQL Community Server (versión 5.7.19). Para la realización de pruebas y llamadas a la API se ha utilizado Postman (versión 5.5.0) y la extensión de Firefox RESTClient.

Las instrucciones a seguir para preparar todo el entorno de desarrollo de manera manual son las siguientes:

1. Instalar PyCharm Community.

<https://www.lifewire.com/how-to-install-the-pycharm-python-ide-in-linux-4091033>
(Instrucciones para instalar PyCharm en Linux)

2. Instalar Python y git:

```
sudo apt-get install python2.7  
sudo apt-get install git (alternativa: sudo apt-get install git-all)
```

3. Instalar el gestor de paquetes pip para Python:

```
sudo apt-get install python-pip python-dev build-essential  
sudo pip install --upgrade pip  
sudo pip install --upgrade virtualenv
```

4. Para instalar automáticamente las librerías necesarias, utilizar el siguiente comando:

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

5. Instalar MySQL Community Server 5.7.19:

- Acceder a <https://dev.mysql.com/downloads/mysql/> y descargar la version correspondiente.
- Descomprimir e instalar (la contraseña de root debe ser root).
- Una vez instalado, abrir la terminal y escribir "sudo su mysql -uroot -proot" para comprobar que todo está correcto.
- Ejecutar el fichero main.py para la creación de la base de datos.

6. Para comprobar el funcionamiento de la API se puede usar, por ejemplo:

- Postman:

Comandos para instalar Postman en Linux:

```
wget https://dl.pstmn.io/download/latest/linux64 -O postman.tar.gz  
sudo tar -xzf postman.tar.gz -C /opt  
rm postman.tar.gz  
sudo ln -s /opt/Postman/Postman /usr/bin/postman
```

- RESTClient:

Instalar la siguiente extensión de Firefox:

<https://addons.mozilla.org/es/firefox/addon/restclient/>

**** Nota:** Los comandos para comprobar si mysql esta activo y activarlo o desactivarlo son:

```
service mysqld status  
service mysqld stop  
service mysqld start
```

**** Nota2:** Si no se puede conectar la api seguir los pasos de la pagina:

<http://www.zyxware.com/articles/5539/solved-cant-connect-to-local-mysql-server-through-socket-var-run-mysqld-mysqld-sock>

Las instrucciones para la integración todos los sistemas se encuentra en la [Wiki de Integración](#).

Gestión del cambio, incidencias y depuración

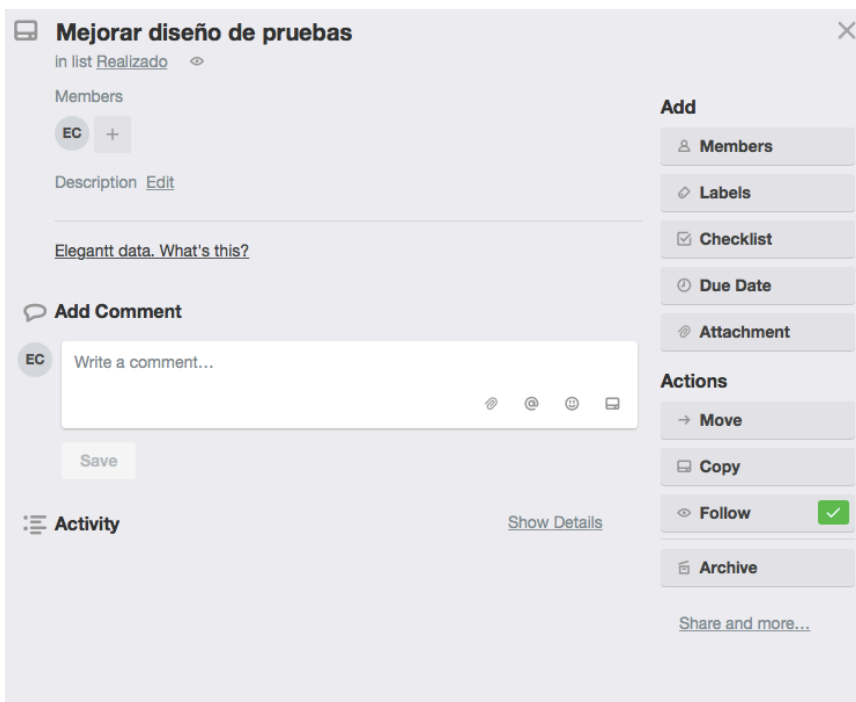
Incidencias Internas

Las incidencias internas se han gestionado principalmente por nuestro grupo de Telegram y mediante el gestor de tareas (nuestro tablero en Trello). Si se trataba de una duda o una petición pequeña simplemente se solucionaba mediante Telegram, en caso contrario, el coordinador añadía una tarea a Trello con la incidencia y asignaba al miembro o miembros del grupo que debían encargarse de solucionarlo. Opcionalmente se indicaba la fecha límite para solucionar la incidencia y la prioridad de la misma.

Ejemplos de incidencias internas:

Incidencia en tarea de Trello:

<https://trello.com/c/T2BSPKKB/39-mejorar-diseño-de-pruebas>



Commit relacionado:

<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/c45efd907df2af98d4dbee71b6ebdb73cb40e1e9>

Corecciones pruebas.py

fix: Arreglos pruebas

Incidencia en tarea de Trello:

<https://trello.com/c/fatpQXP0/27-cambios-en-la-api>

Cambios en la API
×

in list **Realizado**

Members Due Date

+ ✓ Dec 20, 2017 at 12:00 PM

Description [Edit](#)

Elegantt data. What's this?

☒ **Cambios a realizar** [Hide completed items](#) [Delete...](#)

100%

☒ *Inclusión de tokens*
☒ *Cambiar tipo de dato de las respuestas a String*
☒ *Prueba del método de almacenar voto de tal manera que se pueda insertar en la BD todas las respuestas de una misma votación*
Add an item...

Add Comment

EC
Write a comment...

Save

Activity [Show Details](#)

Add

Members

Labels

☒ Checklist

Due Date

Attachment

Actions

→ Move

Copy

Follow

Archive

[Share and more...](#)

Commits relacionados:

<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/20ec56a75457a424cc97a08f3f485580077ead17>

<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/985cc537a1277e3415627ebca58cbb36ae92600a>

<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/2b3a6aa9a2e9ec525a4c0bbc7ce60f3acb3701ac>

Comprobación de tokens ...	20ec56a
martus13 committed 27 days ago add: Terminada la comprobacion de los tokens de todas las llamadas de la api	
Cambios api ...	985cc53
martus13 committed 27 days ago add: añadida la comprobación de los tokens en las llamadas de la api. Merge branch 'master' of https://github.com/jesvazarg/EGC-Almacenamiento-G2 into api	
Cambios api ...	2b3a6aa
martus13 committed 27 days ago fix: correccion de errores y cambio de variables	

Incidencias Externas

Las incidencias externas se han gestionado mediante el gestor de incidencias que ofrece GitHub. Se nombró a un miembro del grupo como gestor de incidencias, el cual debía

informarnos siempre que se produjera una incidencia en nuestro repositorio, poner incidencias en otros repositorios y encargarse de contestar en caso necesario.

Ejemplo de incidencias externas:

Incidencia:

<https://github.com/EGC-G2-Trabajo-1718/almacenamiento/issues/4>



jucansu commented on Dec 13, 2017

Member

Prioridad: Alta

Se necesita saber que método de cifrado de datos va a realizar el equipo de Almacenamiento, para saber como se podrá descifrar los datos necesarios para realizar el recuento de las votaciones.

pedrosr14 commented 27 days ago

Member

Hola, ante todo perdón por la tardanza.

Aún estamos en pruebas viendo qué método vamos a usar porque estábamos usando implementando un método de cifrado que no permitía el descifrado después fácilmente. Ahora mismo estamos tratando de implementar el cifrado AES, cuando lo tengamos se os proporcionará el método y la clave pública que se necesita para descifrar.

pedrosr14 commented 10 days ago • edited

Member

Buenas, el método para descifrar el cifrado que hemos implementado es el siguiente en python

```
def decrypt(texto_encriptado, clave):
    texto_encriptado = base64.b64decode(texto_encriptado)
    iv = texto_encriptado[:AES.block_size]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(texto_encriptado[AES.block_size:])).decode('utf-8')
```

El equipo de recuento necesitaba saber nuestro método de encriptación para poder descifrarlo posteriormente. Una vez implementado nuestro método de cifrado se les comunicó inmediatamente.

Commit relacionado:


<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/1c18f555752937ba62a35382412453d2cc4ddd53>

Cifrado implementado

add: Archivo python con los métodos de cifrado

Incidencia: <https://github.com/EGC-G2-Trabajo-1718/almacenamiento/issues/5>

Definir dependencias en un fichero requirements.txt #5

 **Closed** robgc opened this issue 28 days ago · 1 comment



robgc commented 28 days ago

Owner



Prioridad: Alta

Con objeto de facilitar las tareas de despliegue es importante definir todas las dependencias que vuestro subsistema necesita en un fichero requirements.txt. Aquí tenéis un enlace de referencia:
<https://rukbottoland.com/blog/como-instalar-paquetes-python-con-requirements.txt/>



jesvazarg commented 27 days ago

Member



Muchas gracias, ya está hecho.



jesvazarg closed this 27 days ago

El equipo de integración nos pidió un fichero de requerimientos para automatizar la instalación de las librerías de pip necesarias.

Commit relacionado:

<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/3064a578c0358fa46fca6145dbd560b5c14f8201>

Fichero de dependencia y actualización del manual de instalación

docs: Automatizar la instalación de los paquetes necesarios

Gestión del código fuente

La gestión del código fuente se ha realizado a través de GitHub.

Gestión de commits

Diferenciamos los siguientes tipos de commits:

- **add:** Se añade nueva característica.
- **fix:** Se soluciona un bug.
- **docs:** Se realizaron cambios en la documentación.
- **test:** Se añadieron pruebas.

El formato general de los commits es el siguiente:

Título: Título del commit.

Descripción: Tipo de funcionalidad (una de las mencionadas arriba) y breve resumen.

Ejemplos de commits:

add:

<https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/33720eb258e6d26f293a3e97a260a2293be7917e>

Añadida nueva llamada

add: Se ha añadido una nueva llamada "almacenar_voto_multiple" que recibe un array con los votos del usuario. FALTA COMPROBAR QUE FUNCIONE BIEN.

En este ejemplo se utiliza "add" porque se añade una nueva funcionalidad.

fix: <https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/2b3a6aa9a2e9ec525a4c0bbc7ce60f3acb3701ac>

Cambios api

fix: correccion de errores y cambio de variables

En este ejemplo se usa "fix" ya que se arreglaron errores en la API.

docs: <https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/7e2be2d92fbe6853ffee7b6962c84bf0aca2af85>

Documento de pruebas

docs: Subido el documento de pruebas con portada y formato pdf.

Aquí se usa "docs" ya que se subió un documento relativo a las pruebas, no hubo cambios en el código.

test: <https://github.com/jesvazarg/EGC-Almacenamiento-G2/commit/6d0ca959123994033e9f22e2d25318af061a659d>

Pruebas API

test: Pruebas para los metodos comprobar voto y comprobar voto pregunta.

En este ejemplo se usa “test” porque se han añadido dos pruebas.

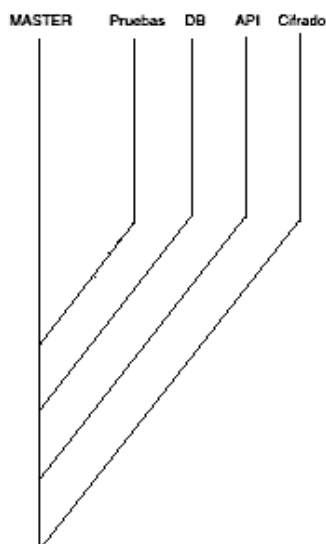
Usage Model

Tenemos un repositorio local en el cual cada usuario crea una rama de desarrollo para realizar las modificaciones que considere necesarias en función de las tareas que les hayan sido asignadas, cuando finalicen sus respectivas tareas se realizara un merge desde las distintas ramas de desarrollo a la rama master. Adicionalmente, también contaremos con una rama de pruebas para testear el código completo antes de cada milestone. Una vez se haya comprobado el correcto funcionamiento del subsistema se realizará un push al repositorio oficial para integrarlo con los demás subsistemas.

Las ramas que hemos utilizado son:

- **master:** Rama principal.
- **db:** Cambios relativos a la base de datos.
- **cifrado:** Rama para implementar código relativo a la encriptación de datos.
- **api:** Cambios relativos a la API.
- **pruebas:** Rama para implementar pruebas.

Esquema de ramas:



Gestión de la construcción e integración continua

Para la construcción e integración continua hemos usado Travis CI tal y como hemos visto en clase. Esta herramienta te permite estructurar un poco el código de manera que quede prueba tu código para verificarlo.

Se ha desplegado nuestro código procedente a nuestro repositorio de la rama master una vez hemos finalizado el proyecto y nuestro código ha pasado la prueba que realiza Travis. Para ello hemos elaborado un fichero `.travis.yml` tal y como se muestra a continuación:

language: python

sudo: required

python:

- "2.7.12"

services:

- mysql

install:

- pip install -r requirements.txt

before_install:

- mysql -e 'DROP DATABASE IF EXISTS almacenamiento'

- mysql -e 'CREATE DATABASE almacenamiento;'

- mysql -u root --default-character-set=utf8 almacenamiento < almacenamiento-votos.sql

- mysql -u root --default-character-set=utf8 almacenamiento < datos-prueba.sql

before_script:

- python main.py

script:

- py.test pruebas.py

Gestión de liberaciones, despliegue y entregas

Política de nombrado e identificación de los entregables

En cuanto a nombrado de archivos, el fichero principal lo llamamos “main.py” y al fichero que gestiona la base de datos lo llamamos “database.py”. El archivo main.py es el que se encarga de lanzar el proceso principal para ejecutar los métodos de la API y también realiza la conexión con la base de datos.

Por otro lado, los commits relativos a cada milestone están identificados en el repositorio oficial por el nombre necesario.

Proceso definido para las entregas

Al acabar cada iteración se realiza un merge de las distintas ramas a la rama master, se realizan las pruebas pertinentes y, si todo está correcto, se hace un push en el repositorio oficial con los demás subsistemas. Posteriormente se realizan las actualizaciones necesarias en la documentación de la wiki, y se realiza una reunión de cierre para revisar que se han cubierto todos los puntos.

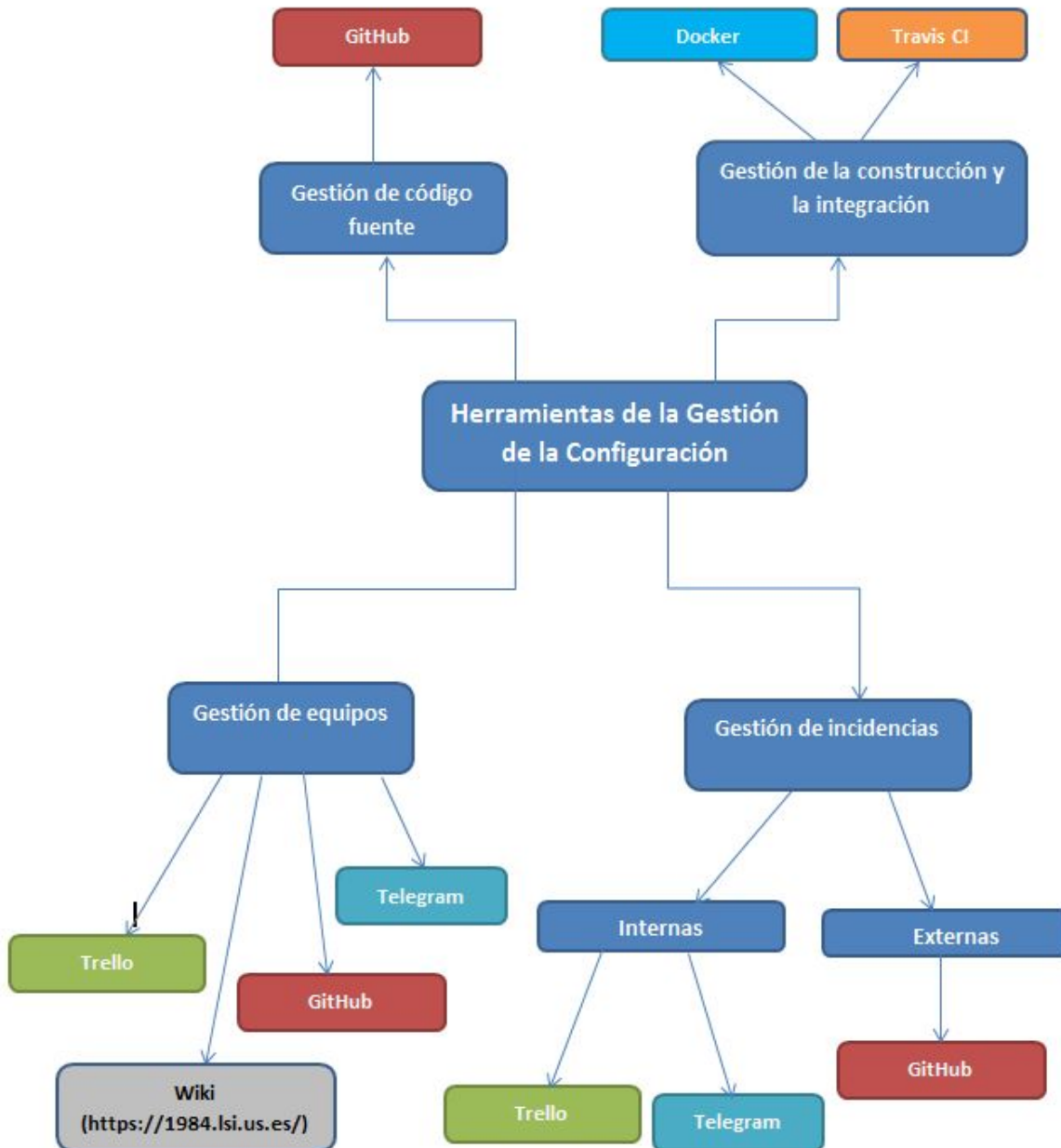
Proceso definido para liberaciones y despliegue

En la entrega final, además de lo mencionado anteriormente, se añadió el archivo de Travis CI para la integración continua, y se realizó el despliegue del proyecto.

Finalmente, en nuestro espacio del portal de Opera, se añadió en la descripción los enlaces a los dos repositorios empleados a lo largo del proyecto, el interno y el oficial integrado.

Mapa de herramientas

A continuación se muestra el mapa de herramientas utilizadas:



Ejercicio de propuesta de cambio

Como propuesta de cambio, hemos optado por implementar un nuevo método en la API que, dado el token de una votación, devuelva los votos de todos los usuarios que han contestado.

El protocolo a seguir será el siguiente:

- 1) Crear issue interna en GitHub. Se creará una incidencia indicando la creación de un nuevo método y se le asignará a todos los miembros del equipo. También se añadirá el reparto de tareas en nuestro tablero de Trello.
- 2) Realizar los cambios pertinentes en la wiki de nuestro grupo. En el apartado de API se deberá describir el nuevo método, indicando los parámetros de entradas, ejemplos de petición, respuestas etc...
- 3) Cambios en el código. Habrá que realizar cambios en `main.py`, donde se implementará un nuevo método `get` llamado `obtener_votos_votacion`. Posteriormente, en `database.py`, se creará el método `consultar_votos_votación` para realizar la consulta a la base de datos. Cada cambio deberá realizarse en la rama pertinente.
- 4) Implementación de pruebas. En el archivo `pruebas.py` se añadirá una prueba funcional para comprobar el correcto funcionamiento de `obtener_votos_votacion` y una prueba unitaria para comprobar el correcto funcionamiento de `consultar_votos_votación`. Estos cambios deben realizarse en la rama de pruebas.
- 5) Modificar documento de pruebas. El documento de pruebas debe incluir la descripción de las dos nuevas pruebas añadidas.
- 6) Se realizará el merge desde todas las ramas que se han usado a la rama master y se realizará un push al repositorio oficial para que se integre con los demás.
- 7) Se despliega el proyecto.

Conclusiones y trabajo futuro

Este trabajo nos ha ayudado a aprender a gestionar la evolución de un proyecto que es parte de un proyecto mayor y por tanto depende otros subsistemas para poder funcionar correctamente. Consecuentemente hemos tenido que adaptarnos a otros equipos y estar en comunicación continua con ellos para que la integración funcione adecuadamente. Hemos aprendido a organizarnos mejor a la hora de trabajar en equipo, así como la importancia de la comunicación interna y de un correcto reparto de tareas para sacar adelante el proyecto cumpliendo los milestones establecidos. También hemos aprendido ciertas herramientas software que desconocíamos previamente, tales como los contenedores de Docker o las herramientas de integración continua para automatizar la preparación del entorno de desarrollo.

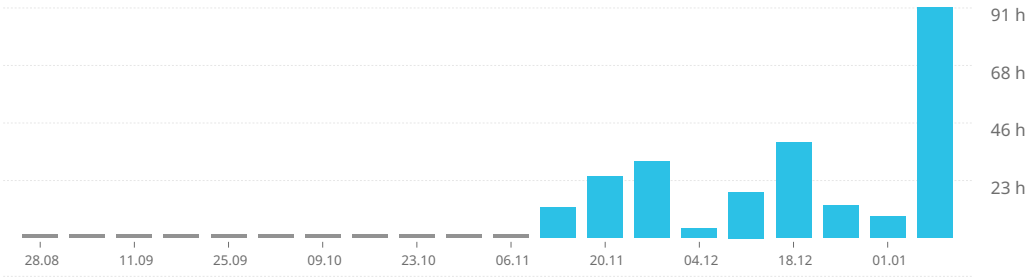
Aunque hemos cumplido con los requisitos del proyecto, hay algunos puntos que se podrían mejorar. Sería recomendable, por ejemplo, buscar un framework para la realización de pruebas para que los cambios realizados por las pruebas en la base de datos no afecten a la base de datos original al finalizar dichas pruebas. Actualmente se ha gestionado esto de manera manual en nuestro script de pruebas.

También podríamos añadir algunos métodos nuevos para que nuestra API sea más eficiente, como por ejemplo devolver todos los votos de una votación en concreto, o devolver todos los votos de un votante específico. Aunque los demás equipos no nos han solicitado cambios adicionales por lo tanto no sería estrictamente necesario.

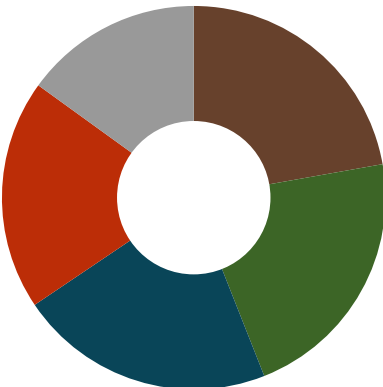
Anexo: Informe de Toggl

Summary report

2017-09-01 - 2018-01-14
Total 235 h 38 min

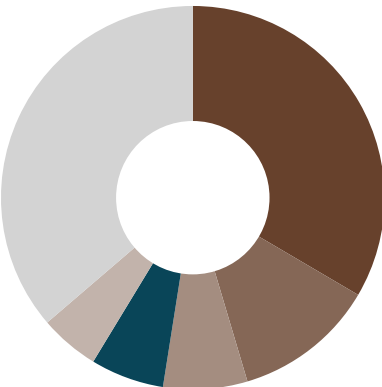


Users



- Elena Camero 52:19:06
- Marta Carmona 51:23:09
- Jesús 50:47:06
- Jose Luis Martinez 45:43:44
- Pedro Serrano 35:25:52

Time entries



- Reunión de planificación 78:50:36
- Finalización del proyecto 28:08:22
- Realización de pruebas 16:43:19
- Creación de Docker 14:35:13
- Elaboración Documento del... 11:51:00
- Other 85:30:27

Users / Time entries	Duration
Elena Camero	52:19:06
Correcciones Pruebas	3:03:00
Correcciones Wiki	3:50:58
Documentación de la API	1:22:00
Elaboración de la presentación PechaKucha	2:55:00
Elaboración Documento del Proyecto	11:31:00
Elaboración fichero de dependencia	0:54:08
Estructura de la Wiki	1:22:00
Finalización del proyecto	5:33:00
Realización de pruebas	4:17:00
Reunión de planificación	15:44:00
Revisión de requisitos	0:20:00
Revisión Trabajo	1:27:00
Jesús	50:47:06
Cambios en la BD	4:20:00
Corrección de pruebas	0:45:00
Creación de la BD	3:43:00
Crear actas de reunión	3:21:10
Crear ordenes del día	1:44:00
Crear una nueva rama	0:33:00
Elaboración de la presentación PechaKucha	2:55:00
Elaboración Documento del Proyecto	0:20:00
Entrega proyecto	0:23:00
Estudios de los requisitos	1:13:47
Finalización del proyecto	5:33:00
Instalación del software	1:03:00
Planificación de tareas	1:04:09
Realización de pruebas	4:08:00
Reunión de coordinadores	1:00:00
Reunión de planificación	15:44:00
Revisar trabajos	2:57:00
Jose Luis Martinez	45:43:44
Creación de Docker	14:35:13
Crear plantillas de documentos	1:07:03
Elaboración de la presentación PechaKucha	2:55:49
Finalización del proyecto	6:23:09
Instalación del software	0:46:38
Realización de pruebas	4:03:51

Reunión de planificación	15:52:01
Marta Carmona	51:23:09
Buscar un framework para pruebas	5:00:00
Cambios en la API	3:25:18
Cambios en la API - Cambiar tipo de dato de las respuestas a String	0:24:21
Cambios en la API - Inclusión de tokens	1:30:00
Cambios en la API - Prueba del método de almacenar voto de tal manera que se pueda insertar en la BD todas las respuestas de una misma votación	4:05:27
Creación de Docker y Travis	3:54:00
Creación de la API	4:09:05
Creación del manual de instalación	0:52:04
Finalización del proyecto	6:12:00
Realización de pruebas	4:14:28
Reunión de planificación	15:45:25
Revisión trabajo	0:20:01
Reunión de coordinadores	1:31:00
Pedro Serrano	35:25:52
Arreglos en el cifrado	0:44:37
Arreglos en la API	0:56:15
Cambios en la API - Cifrado	0:32:33
Corrección de pruebas	0:43:02
Documento de pruebas	2:34:18
Finalización del proyecto	4:27:13
Implementación del encriptado	2:38:14
Instalación de la máquina virtual y el material necesario	2:00:07
Investigación del método de encriptado	2:09:58
Mapa de herramientas	0:52:34
Preparación del entorno de PyCharm	0:26:01
Pruebas	0:38:49
Reunión de planificación	15:45:10
Revisión de la API	0:57:01

Created with toggl.com