Ingeniería informática

Universidad Politécnica de Madrid

Facultad de Informática / Escuela Técnica Superior de Ingenieros Informáticos

Proyecto fin de carrera:

Sistema de negocio de cambios de grupo

Autor: Rafael Mozún Villamayor
Tutora: Angélica de Antonio Jiménez

Madrid, Julio 2018

Este trabajo no hubiera sido posible sin Angel Luis Sancho Praena, compañero en tantas prácticas durante los años en la facultad. Mi agradecimiento también para Felipe Madroñal y Javier Diez “Cholo”, que me han enseñado, sobre todo, cosas no técnicas. Gracias a mi familia, a mi hermano David García Puras y, todos los días, a Pam, Larry, Sam y Mel.

“Gabba gabba hey!”

“And in the end
The love you take
Is equal to the love you make”

Índice de contenidos

1- Introducción

Este “Sistema de Negocio de Cambios de Grupo” surge de una necesidad que tiene la jefatura de estudios de la Escuela Técnica Superior de Ingenieros Informáticos (antes Facultad de informática) de la Universidad Politécnica de Madrid.

Actualmente los alumnos se matriculan en la escuela mediante un sistema que les asigna aleatoriamente los grupos de las asignaturas en las que se matriculan. Al ocurrir esto es muy habitual que un importante porcentaje de alumnos soliciten cambios de grupo en el mismo momento en el que finalizan su matriculación. Dichas solicitudes de cambio de grupo han sido gestionadas hasta ahora manualmente por jefatura de estudios con la gran carga de trabajo que esto requiere.

La entonces jefa de estudios Angélica de Antonio Jiménez propone realizar una aplicación web que gestione las solicitudes de cambio de grupo y pueda dar una solución eficiente a la problemática presentada. Este proyecto fin de carrera trata sobre el análisis, implementación y pruebas de esta aplicación web.

El capítulo 2 “Tecnologías y modelo arquitectónico empleado” detalla qué tecnologías se han empleado y en qué versiones o implementaciones. El capítulo 3 contiene el “Diseño de alto nivel”, con los casos de uso en formato simple y extendido, así como el modelo de dominio y los diagramas de secuencia de sistema. En el siguiente capítulo se detalla el diseño de bajo nivel con las clases y estructuras de base de datos.

El capítulo 5 detalla los pasos seguidos durante la implementación, sus correspondientes pruebas y las dificultades encontradas. El capítulo 6 “Implantación” sirve como manual de instalación y uso, mientras que en el 7º constatamos las conclusiones y planteamos las posibles líneas futuras de este proyecto. Por último tenemos el capítulo 8 con las referencias utilizadas.

2- Tecnologías y modelo arquitectónico empleado

A la hora de elegir las tecnologías que utilizase este sistema se preguntó al departamento responsable del futuro mantenimiento del sistema, el centro de cálculo de la ETSI Informáticos, que recomienda Laravel, ya que es la tecnología que usa para otras cuestiones y así sería más sencillo para ellos la futura implantación y mantenimiento.

Laravel es un *framework* de código abierto para desarrollar aplicaciones y servicios web con PHP 5 y PHP 7. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti". Fue creado en 2011 y tiene una gran influencia de *frameworks* como Ruby on Rails, Sinatra y ASP.NET MVC.2

2.1- Laravel y la filosofía MVC

Laravel, aunque en puridad no es una arquitectura MVC (Modelo Vista Controlador), tiene todas las facilidades de la misma a la hora de agrupar componentes en esas categorías, de hecho hasta Laravel 4 se incluían las carpetas *controllers*, *models* y *views*, en Laravel 5 la estructura de carpetas es ligeramente diferente, conteniendo los modelos en la carpeta app, los controladores en app/http/controllers y las vistas en resources/views.

La filosofía MVC va cambiando con el tiempo, y en el caso que nos ocupa de Laravel el propio creador del *framework* Taylor Otwell, ha declarado que “no hay manera de encapsular todos los aspectos de aplicaciones web robustas en esas 3 letras.”. Los conceptos modelo vista y controlador son demasiado genéricos y distan muchísimo de proporcionar las herramientas que se necesita para construir aplicaciones avanzadas.

Al utilizar Laravel 5 sí que podemos usar una serie de ventajas que una arquitectura MVC estricta no tiene, por ejemplo la manera de representar las rutas que se encargan de analizar las URLs y asignarlas a un método o función, o también un middleware que restrinja ciertas áreas de la aplicación a una serie de usuarios.

2.2- Arquitectura interna de Laravel

Internamente, la arquitectura de Laravel es un flujo de comunicación entre el *Foundation* del *framework*, los *Services Providers*, una estructura de *Controllers* con Middleware y una capa de servicios que se comunica con el acceso a datos del ORM (*Object-Relational mapping*, o lo que es lo mismo, mapeo de objeto-relacional) y al final, con la base de datos.

El *Foundation* es el núcleo del *framework*, los *Services Providers* son la base del inicio de la aplicación, la arrancan y es donde está la conexión entre el *Foundation* de Laravel y las rutas de la app, interfaces, objetos de servicios, contenedores, eventos, errores, etc.

Los *Middleware* se refiere a los objetos que cubren las cosas que siempre se necesitan en una aplicación web: cifrado de cookies, autenticación de usuarios, protección contra vulnerabilidades como CSRF (*cross site request forgery*) y XSS (*cross-site scripting*), etc. El ORM es el acceso a la base de datos y se pueden realizar consultas a través de ORM sin escribir SQL, además estas consultas están ya optimizadas para que sean más eficientes y además no haya que preocuparse de ataques SQL injection.

2.3- Versión de Laravel

En este sistema se ha utilizado la versión 5.2.45 de Laravel, que es la que en el momento de inicio de la implementación era la versión estable recomendada. Su fecha de publicación es el 26 de agosto de 2016 y su código se encuentra en la url: https://github.com/laravel/framework/releases/tag/v5.2.45

2.4- Entorno de desarrollo

El entorno de desarrollo utilizado en la implementación ha sido “Homestead” el entorno de desarrollo oficial de Laravel. Homestead es una "Box de Vagrant". Vagrant es una capa por encima de Oracle VM Virtualbox (o de otros sistemas de virtualización como VMWare) que nos sirve para crear entornos de desarrollo y las Box en su terminología son imágenes de sistemas operativos ya instalados, en general vienen con la máquina “vacía”, como si hubiéramos acabado de instalar el sistema operativo en ese momento, pero en el caso de Laravel Homestead, viene con distintos paquetes de software ya configurados, sobre un sistema operativo Ubuntu instala: PHP, HHVM, Nginx, MySQL, PostgreSQL, NodeJS, Redis y algunas librerías como Memcached, Laravel Envoy, Beanstalkd, etc.

Vagrant te permite la creación y gestión sencilla de entornos de trabajo "portables" y replicables que funcionan sobre tecnologías de virtualización conocidas, ofreciendo además un modo de trabajo claro para poder transportar dichos entornos y que funcionen sin problemas en otro lugar: nuevamente en un servidor, la nube, etc... Vagrant se puede asimilar a un gestor de máquinas virtuales ya que por debajo usa la tecnología de virtualización que nos interese: VMWare, VirtualBox, Hyper-V, Amazon web Services, RackSpace Cloud, Google Compute Engine…

Su factor diferencial es lo mucho que facilita la creación de las máquinas virtuales: creas un archivo que describe el tipo de máquina que necesitas, el software que tiene que tener instalado, la forma de acceder a la máquina virtual... Luego abres una línea de comandos, lanzas una instrucción y en un momento Vagrant te crea el entorno completo que necesites,

tal y como lo has descrito. Luego te puedes llevar ese entorno a cualquier otro sistema y usarlo desde allí.

2.5- Otras opciones contempladas

En una primera aproximación al proyecto se estimó su implementación en *Ruby On Rails*, al ser una tecnología muy extendida en desarrollo de aplicaciones web, que se ha mostrado muy eficiente y que tiene una de las mayores comunidades de desarrolladores. Tal y como se relata en el primer párrafo de este capítulo se consultó al centro de cálculo de la ETSI Informáticos y esta vía fue desechada.

3- Diseño de alto nivel (análisis, requisitos, especificación)

En una primera reunión hacemos la toma de requisitos y, a grosso modo, llegamos a estas conclusiones:

Básicamente la aplicación será un “Tablón de ofertas de intercambio de grupos” donde los alumnos podrán:

- Crear una solicitud de cambio de grupo para asignatura/grupo origen/grupo destino
- Ver los cambios que piden otros alumnos
- “Reservar” un cambio que vea que es compatible con el que quiere él

Además, el sistema por sí solo tiene que ser capaz de “cuadrar” cambios.

El alumno se identificará mediante su email, podría hacerlo con su número de matrícula, pero es más útil el email y además en este punto el número de matrícula no es importante, ya que el cambio de grupo en realidad no será efectivo hasta que lo presente en secretaría/jefatura de estudios.

A continuación pasamos a identificar los actores y los casos de uso.

3.1- Casos de uso

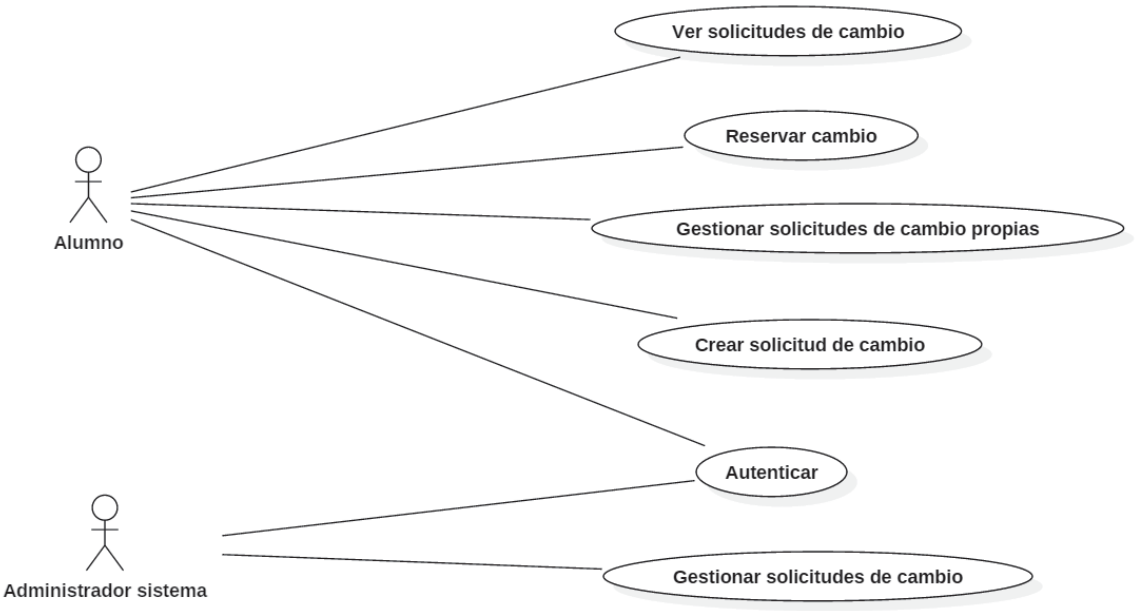
Se identifican dos actores: alumno y administrador del sistema

Los casos de uso identificados para cada actor son los siguientes:

- Alumno:
    - Crear solicitud de cambio
    - Ver solicitudes de cambio
    - Reservar cambio
    - Gestionar solicitudes de cambio propias
    - Autenticar

- Administrador del sistema:
    - Gestionar solicitudes de cambio
    - Autenticar

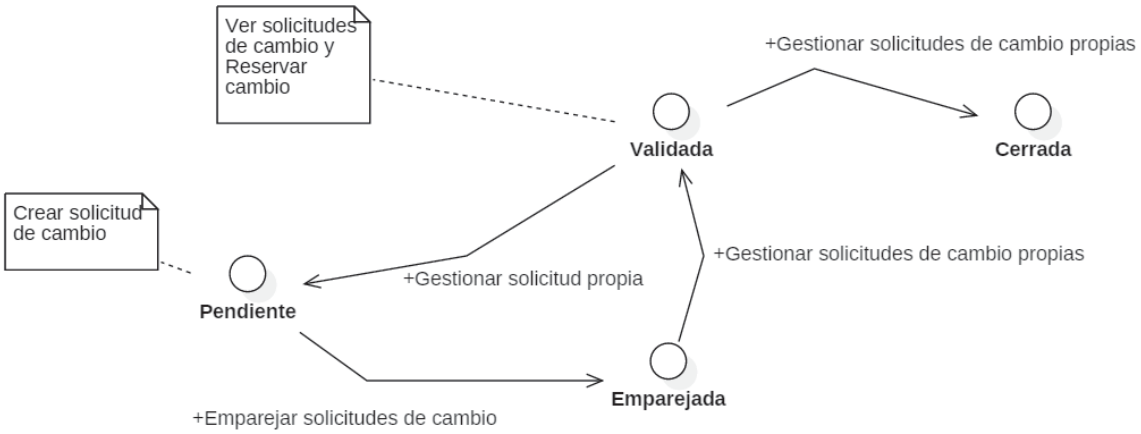
3.2- Diagrama de Casos de Uso



3.3- Diagrama de estados

3.3.1- Solicitud de cambio

Una solicitud de cambio de grupo pasará por diferentes estados, siendo el ciclo de vida de la solicitud el expuesto en el siguiente diagrama:



- Estado Pendiente: solicitud de cambio nueva.
- Estado Emparejada: la solicitud de cambio está asociada a otra que consta de la misma asignatura, diferente alumno y grupos de origen y destino inversos.
- Estado Validada: la solicitud está emparejada y además uno de los dos alumnos ha validado el emparejamiento.
- Estado Cerrada: ambos alumnos han validado el emparejamiento.

=> pendiente
Cuando se crea una nueva solicitud de cambio su estado es pendiente (caso de uso: Crear solicitud de cambio)

=> validada
Cuando se crea una solicitud de cambio a partir de otra preexistente, su estado es directamente validada (caso de uso: Ver solicitudes de cambio + Reservar cambio)

validada => pendiente
Anulación de reserva de cambio por alumno (caso de uso: Gestionar solicitudes de cambio propias)

pendiente => emparejada
Emparejamiento automático por sistema (caso de uso: Emparejar solicitudes de cambio)

emparejada => validada

Validación de reserva de cambio por alumno 1 (caso de uso: Gestionar solicitudes de cambio propias)

validada => cerrada
Validación de reserva de cambio por alumno 2 (caso de uso: Gestionar solicitudes de cambio propias)

Al llegar al estado Cerrada ya existe un acuerdo de intercambio validado por ambas partes, quedando pendiente únicamente su formalización en secretaría o donde proceda.

3.3.2- Bloque de solicitudes

El bloque de solicitudes es cuando el usuario solicita varios cambios de grupo pero quiere que se le asignen todos ellos, sin cambios parciales. Es por esto que una solicitud de cambio que pertenezca a un bloque sólo podrá cerrarse (pasar a estado Cerrada) si todas las solicitudes de dicho bloque están en estado Validada.

3.4- Casos de uso en formato extendido

UC1: Crear solicitud de cambio
El Alumno crea una solicitud de cambio de grupo indicando la asignatura, grupo origen y grupo destino.
Actor: Alumno
PRE: El usuario está autenticado
PRE: El Alumno está matriculado en la escuela (por lo que tiene número de matrícula) y en la asignatura y grupo de origen que indica.
POST: El sistema registra la solicitud del alumno

1- El alumno pide crear una solicitud de cambio

2- El sistema ofrece opciones de asignaturas y grupos del plan de estudios para crear una solicitud de cambio

3- El alumno selecciona una asignatura, dos grupos (origen y destino) y si forma bloque o no para crear una solicitud de cambio

4- El sistema registra la nueva solicitud del alumno

5- El sistema notifica vía email al alumno que su solicitud ha sido registrada con éxito

6- El sistema intenta emparejar la nueva solicitud con las que tenga pendientes en ese momento

UC2: Ver solicitudes de cambio
El Alumno puede ver los cambios que han solicitado otros alumnos.
Actor: Alumno
PRE: El usuario está autenticado

1- El alumno pide ver las solicitudes de cambio

2- El sistema muestra las solicitudes en estado pendiente que están registradas en el sistema

UC3: Reservar cambio
El Alumno reserva una solicitud de cambio existente porque él quiere solicitar el inverso, es decir, idéntica asignatura y grupos origen y destino en orden inverso.
Actor: Alumno
PRE: El usuario está autenticado
PRE: El alumno tiene las solicitudes pendientes de otros alumnos visibles

PRE: Existe una solicitud de cambio de otro alumno que coincide en asignatura y cuyos grupos orígenes y destino son los inversos a los deseados por el Alumno que ejecuta este caso de uso.
POST: El sistema crea una nueva solicitud de cambio que se emparejará directamente con la solicitud a partir de la cuál fue creada. Además la solicitud pasa directamente al estado validada, ya que dicho emparejamiento estará validado por el alumno que reserva el cambio.

1- El alumno pide reservar un cambio

2- El alumno selecciona una solicitud de cambio existente de otro alumno para crear una solicitud de cambio inversa (es decir, misma asignatura y grupos destino y origen intercambiados) y si quiere que forme bloque con las otras solicitudes que tenga en el sistema

3- El sistema registra la nueva solicitud del alumno

4- El sistema notifica vía email a ambos alumnos del nuevo emparejamiento creado

UC4: Gestionar solicitudes de cambio propias
El Alumno podrá eliminar solicitudes de cambio o bien aceptar emparejamientos que hayan sido propuestos por otro Alumno o bien por el propio sistema.
Actor: Alumno
PRE: El usuario está autenticado
PRE: Existen solicitudes de cambio activas en el sistema para el alumno

1- El alumno pide gestionar sus solicitudes de cambio

2- El sistema muestra las solicitudes del alumno que están registradas en el sistema. (Si una solicitud está en estado cerrada no podrá efectuar ninguna acción sobre ella)

3- Usuario pide desemparejar solicitud en estado validada (volverá a estado pendiente)
3a- Usuario pide cerrar solicitud en estado validada (pasará a estado cerrada)
3b- Usuario pide validar solicitud en estado emparejada (pasará a estado validada)
3d- Usuario pide modificar su configuración de bloque. Un usuario sólo puede establecer que sus solicitudes funcionen como bloque cuando todas ellas están en estado pendiente.

4- El sistema registra el cambio en la solicitud del alumno

5- El sistema notifica vía email al alumno/s del cambio de estado de las solicitudes

6- El sistema intenta emparejar solicitudes pendientes con los cambios realizados

UC5: Autenticar

El Alumno o Administrador se autentica en el sistema.
Actor: Alumno y Administrador del sistema
POST: El actor (Alumno o Administrador del sistema) que invoca el caso de uso está autenticado

1- El sistema pide al usuario los datos para autenticarse

2- El usuario proporciona al sistema sus credenciales de acceso

3- La autenticación tiene éxito y el sistema ofrece al usuario un menú con todas las opciones disponibles

3a- La autenticación no tiene éxito, el usuario no puede acceder al sistema

UC6: Gestionar solicitudes de cambio
El Administrador podrá gestionar todas las solicitudes, tantos las pendientes como las aceptadas, y cambiar tanto sus datos como su estado siempre que se cumplan las condiciones generales para un emparejamiento.
PRE: El usuario está autenticado

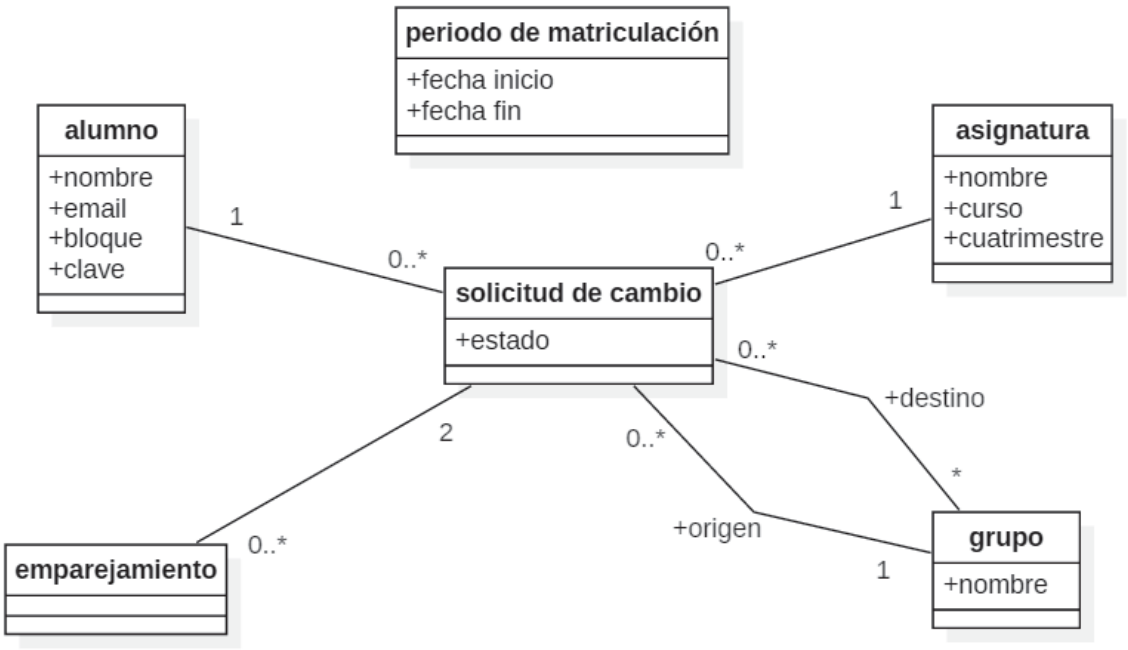
1- El sistema muestra todas las solicitudes registradas en el sistema, además de la posibilidad de modificar la fecha límite de creación de solicitudes y de resetear el sistema.

2a- El administrador pide cambiar el estado de una solicitud (que puede implicar el cambio de estado de otra)
2b- El administrador pide crear una solicitud de cambio pendiente, seleccionando un usuario, una asignatura y dos grupos
2c- El administrador pide crear una solicitud de cambio emparejada, seleccionando dos usuarios, una asignatura y dos grupos
2d- El administrador pide crear un emparejamiento seleccionando dos solicitudes existentes en estado pendiente

3- El sistema registra el cambio en la solicitud del alumno.

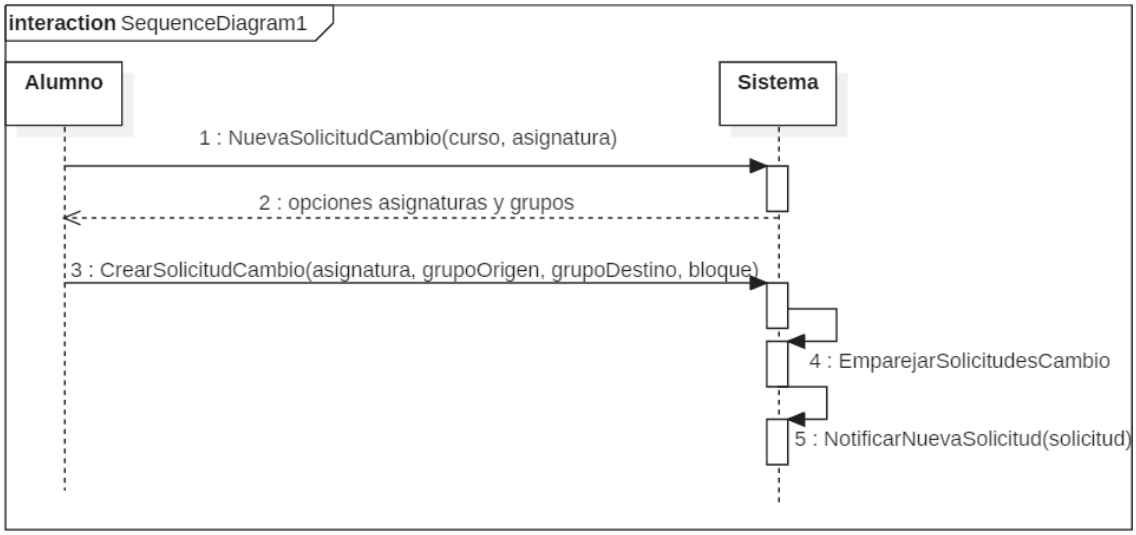
4- El sistema notifica vía email al alumno/s del cambio de estado de las solicitudes

3.5- Modelo de Dominio



3.6- Diagramas de secuencia de sistema y contratos de operación

UC1: Crear solicitud de cambio



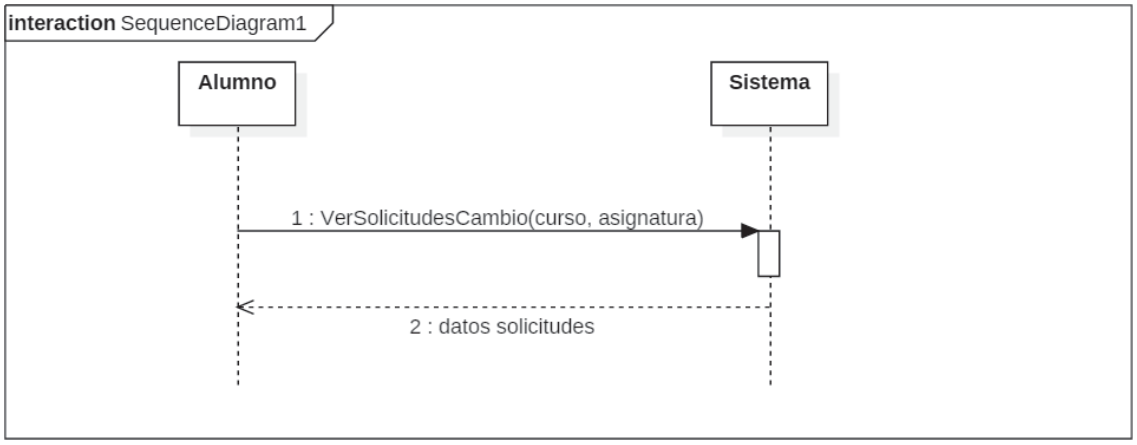
1- NuevaSolicitudCambio(curso, asignatura)
PRE: El usuario está autenticado
POST: ninguna
Responsabilidades: El sistema devuelve las opciones de asignaturas y grupos disponibles para crear una solicitud de cambio

3- CrearSolicitudCambio(asignatura, grupoOrigen, gruposDestino, bloque)
POST: El sistema crea una instancia de solicitud de cambio con su atributo estado a “pendiente”. El sistema crea nuevas asociaciones entre la nueva solicitud creada, el alumno que la crea y la asignatura y los grupos que correspondan.
Responsabilidades: El sistema registra la nueva solicitud del alumno y lanza los procesos de emparejamiento y notificación.

4- EmparejarSolicitudesCambio()
Responsabilidades: El sistema intenta emparejar la nueva solicitud con las que tenga pendientes en ese momento
POST: Si existe una solicitud de cambio inversa a la creada el sistema empareja dichas solicitudes

5- NotificarNuevaSolicitud(solicitud)
Responsabilidades: El sistema notifica vía email al alumno de que su solicitud ha sido registrada con éxito

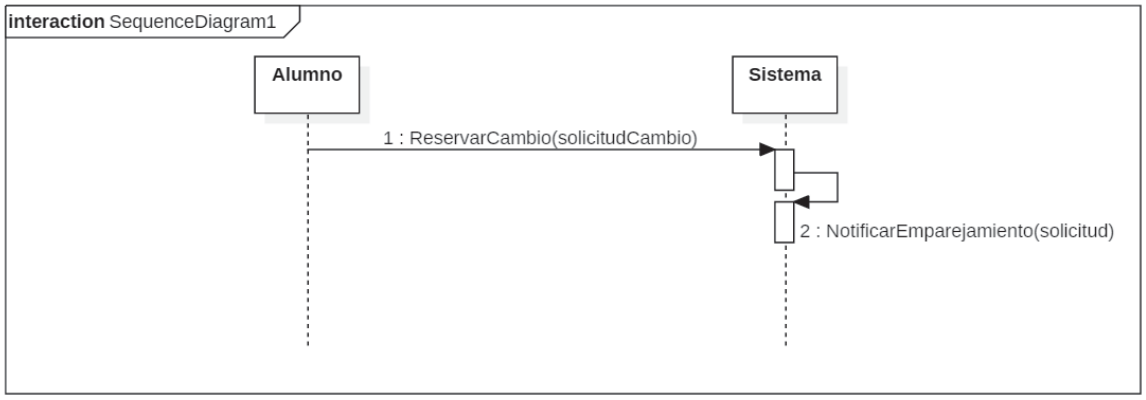
UC2: Ver solicitudes de cambio



1- VerSolicitudesCambio(curso, asignatura)
PRE: El usuario está autenticado
POST: ninguna
Responsabilidades: El sistema devuelve las solicitudes de cambio cuya asignatura y curso sea la introducida y que estén disponibles para ser emparejadas para crear una solicitud de cambio.

1b- VerSolicitudesCambio()
PRE: El usuario está autenticado
POST: ninguna
Responsabilidades: El sistema devuelve todas las solicitudes de cambio que estén disponibles para ser emparejadas para crear una solicitud de cambio.

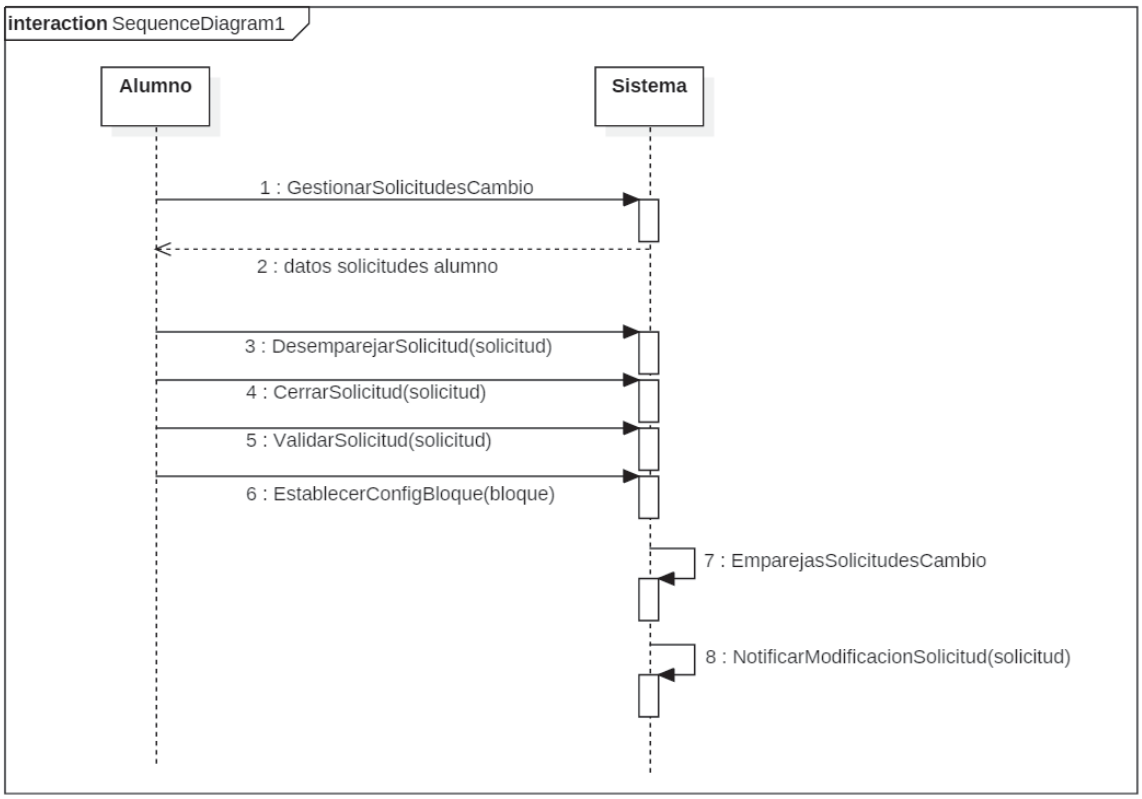
UC3: Reservar cambio



1- ReservarCambio()
PRE: El usuario está autenticado
PRE: El alumno tiene las solicitudes pendientes de otros alumnos visibles
PRE: Existe una solicitud de cambio de otro alumno que coincide en asignatura y cuyos grupos orígenes y destino son los inversos a los deseados por el Alumno que ejecuta este caso de uso.
POST: El sistema crea una instancia de solicitud de cambio con su atributo estado a “validada” (ya que dicho emparejamiento estará validado por el alumno que reserva el cambio). El sistema crea nuevas asociaciones entre la nueva solicitud creada, el alumno que la crea y la asignatura y los grupos que correspondan. El sistema crea una nueva instancia de emparejamiento que asocia a las dos solicitudes. El sistema cambia el atributo estado de la solicitud objeto de la reserva a “emparejada”
Responsabilidades: Lanza el proceso de NotificarEmparejamiento

2- NotificarEmparejamiento(solicitud)
Responsabilidades: El sistema notifica vía email al alumno de que su solicitud ha sido registrada con éxito

UC4: Gestionar solicitudes de cambio propias



1- GestionarSolicitudesCambio()
PRE: El usuario está autenticado
PRE: Existen solicitudes de cambio activas en el sistema para el alumno
Responsabilidades: el sistema muestra las solicitudes del alumno que están registradas en el sistema. (Si una solicitud está en estado cerrada no podrá efectuar ninguna acción sobre ella)

3- DesemparejarSolicitudCambio(solicitud)
PRE: Solicitud está en estado validada
POST: El estado de la solicitud se establece a “pendiente”

3a- CerrarSolicitudCambio(solicitud)
PRE: solicitud está en estado validada
POST: El estado de la solicitud se establece a “cerrada”

3b- ValidarSolicitudCambio(solicitud)
PRE: solicitud está en estado emparejada
POST: El estado de la solicitud se establece a “validada”

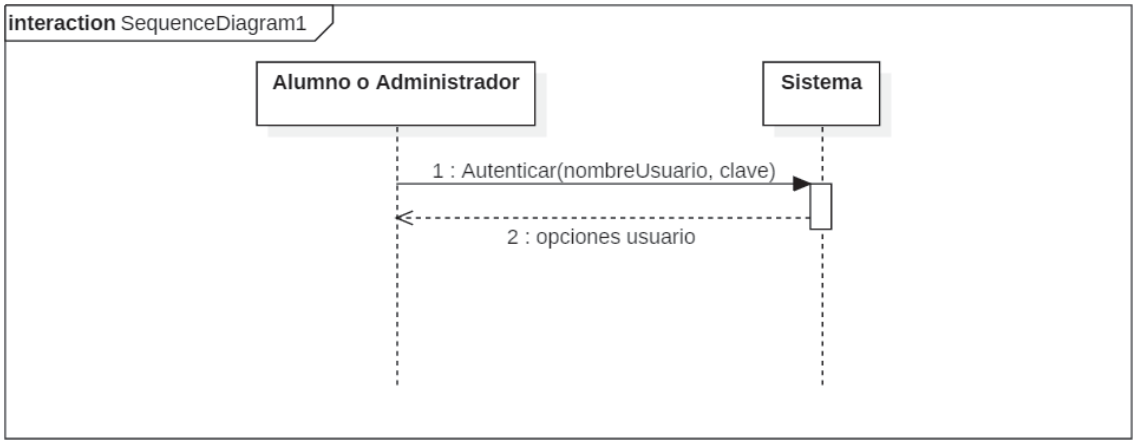
3c- EstablecerConfigBloque(bloque)

POST: Se modifica el valor de la configuración de bloque del usuario. Un usuario sólo puede establecer que sus solicitudes funcionen como bloque cuando todas ellas están en estado pendiente.
Excepción: Si el usuario quiere activar el modo bloque y tiene alguna solicitud no pendiente, entonces el sistema no le deja efectuar la operación y le muestra un mensaje de error

4- EmparejarSolicitudesCambio()
Responsabilidades: El sistema intenta emparejar la solicitud recién modificada con las que tenga pendientes en ese momento
POST: Si existe una solicitud de cambio inversa a la modificada el sistema empareja dichas solicitudes

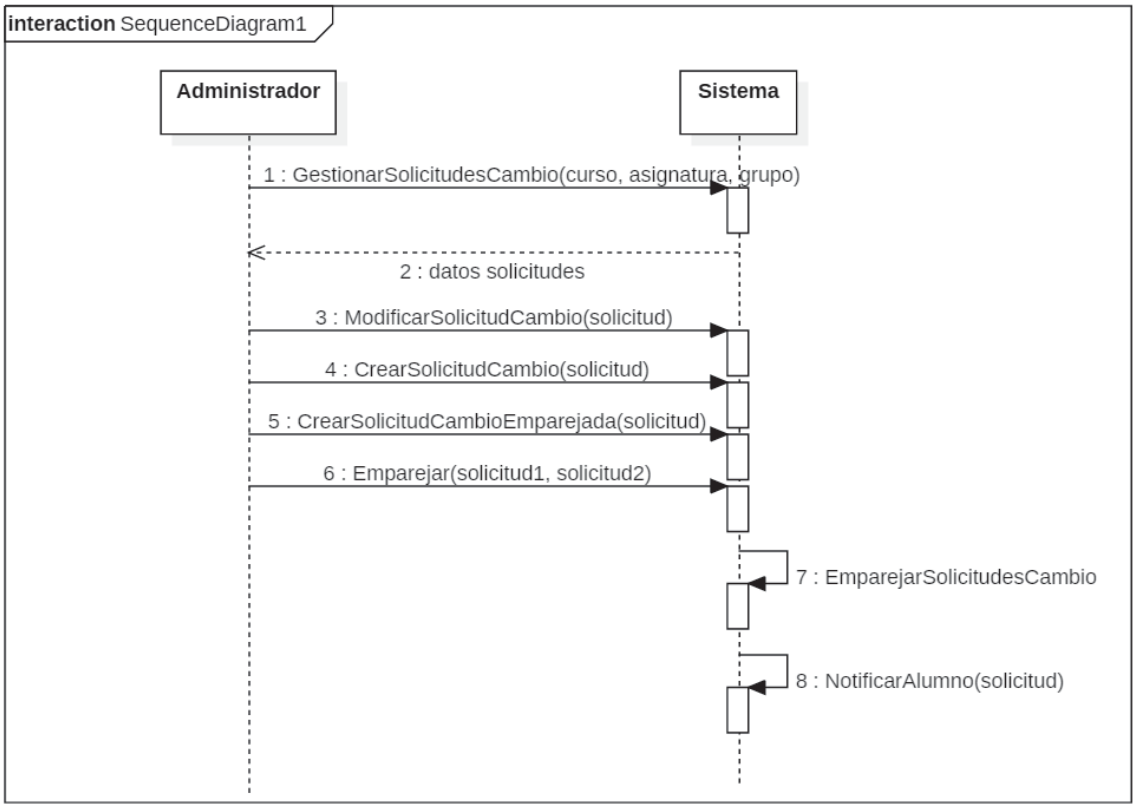
5- NotificarModificacionSolicitud(solicitud)
Responsabilidades: El sistema notifica vía email al alumno de que su solicitud ha sido modificada con éxito

UC5: Autenticar



1- Autenticar(nombreUsuario, clave)
POST: El actor (Alumno o Administrador del sistema) que invoca el caso de uso está autenticado
Responsabilidades: El sistema comprueba nombre de usuario y clave y si es válido le muestra todas las opciones disponibles.
Excepciones: La autenticación no tiene éxito, se muestra mensaje de error y el usuario no puede acceder al sistema

UC6: Gestionar solicitudes de cambio



1- GestionarSolicitudesCambio(curso, asignatura, grupo)
PRE: El usuario está autenticado
Responsabilidades: el sistema muestra todas las solicitudes registradas en el sistema.

3- ModificarSolicitudCambio(solicitud)
POST: El administrador cambia el estado de una solicitud (que puede implicar el cambio de estado de otra). Se puede modificar tanto su estado como sus asociaciones con alumnos.

3a- CrearSolicitudCambio(solicitud)
POST: El administrador crea una nueva solicitud de cambio que tendrá el estado “pendiente” y se asociará a un alumno, una asignatura y dos grupos

3b- CrearSolicitudCambioEmparejada(solicitud)
POST: El administrador crea una nueva solicitud de cambio que tendrá el estado “emparejada” y se asociará a dos alumnos, una asignatura y dos grupos

3c- Emparejar(solicitud1, solicitud2)
POST: El administrador crea un emparejamiento seleccionando dos solicitudes existentes en estado “pendiente”, es decir, dicha solicitud pasa a tener estado “emparejada”.

4- EmparejarSolicitudesCambio()

Responsabilidades: El sistema intenta emparejar la solicitud recién creada o modificada con las que tenga pendientes en ese momento
POST: El sistema empareja el máximo número de solicitudes que pueda

5- NotificarAlumno(solicitud)
Responsabilidades: El sistema notifica vía email al alumno del cambio realizado por el administrador en su solicitud.

4- Implementación y pruebas

Una vez ha quedado concluido el diseño del aplicativo comenzamos con la implementación del código. Aunque ha habido algunas excepciones, el proceso de implementación de la aplicación web ha seguido prácticamente el orden que se detallan en los próximos capítulos.

Empezamos explicando la configuración del entorno de desarrollo empleado, continuamos creando un proyecto básico y explicando, dada su importancia, los directorios y ficheros básicos que nos encontramos en el código Laravel.

Después pasamos a la creación de las migraciones que al ejecutarse dejarán lista la estructura de base de datos. Proseguimos con la creación de los modelos y con la precarga de datos para posibilitar la realización de pruebas.

Más adelante creamos el middleware que servirá para la autenticación, así como las políticas de autorización, que más tarde iremos completando según las necesidades que nos vayamos encontrando.

A continuación una sección dedicada a depuración y errores, explicando las diferentes maneras de trazar la ejecución del código, así como la configuración de aspectos generales de la aplicación como idioma, página genérica de errores de validación, etc.

Para terminar la parte de modelos empleamos la tecnología eloquent, presente en Laravel, para definir las relaciones entre los diferentes modelos y que se cumplan las restricciones del modelo del dominio.

En el siguiente paso decidimos la interfaz gráfica de usuario para que tenga una identidad definida, además de definir algunos detalles de colores e iconos para que las vistas sean lo más intuitivas posible.

Una vez hecho esto podemos empezar el desarrollo de las diferentes características y operaciones presentes en los casos de uso. Seguimos el orden lógico con la creación de una solicitud de cambio, que implica tanto la creación de una vista como las operaciones pertinentes en el controlador.

De forma análoga implementamos la posibilidad de que un usuario pueda ver el listado de las solicitudes disponibles y que pueda reservar alguna de estas solicitudes. Un usuario también podrá consultar el estado de sus propias solicitudes de cambio.

El usuario administrador tendrá privilegios para poder crear solicitudes de cambio de cualquier usuario y además podrá ver todas las solicitudes de cambio existentes.

Para finalizar el desarrollo propiamente dicho, una sección dedicada al *mailing*, es decir, cómo enviamos los correos de información a los usuarios y qué contenido tendrán dichos correos.

Por último un resumen de las pruebas efectuadas a lo largo de todo el proceso de implementación, así como un resumen del ciclo de vida habitual en una solicitud de cambio.

4.1- Entorno de desarrollo

Tal y como hemos comentado anteriormente, para desarrollar hemos utilizado el entorno de desarrollo oficial de Laravel, “Homestead”.

4.1.1- Instalación de Homestead

Antes de lanzar el entorno Homestead instalamos en nuestra máquina de desarrollo Oracle VirtualBox (Versión 5.1.30 r118389 (Qt5.6.2)), así como Vagrant. Una vez tenemos esto añadimos la box de laravel homestead a nuestra instalación de vagrant. Configuramos el archivo Homestead.yaml (que reproducimos más abajo) para que utilice el proveedor de vagrant a usar, en este caso virtualbox. También configuramos el mapeo de directorios.

```
---
ip: "192.168.10.10"
memory: 2048
cpus: 1
provider: virtualbox

authorize: ~/.ssh/id_rsa.pub

keys:
    - ~/.ssh/id_rsa

folders:
#    - map: ~/Code
    - map: C:\Laravel\Code
      to: /home/vagrant/Code

sites:
    - map: pdf.app
      to: /home/vagrant/Code/pfc/public
#     to: /home/vagrant/Code/Quickstart/public

databases:
    - homestead

# blackfire:
#     - id: foo
#       token: bar
#       client-id: foo
```

```
#      client-token: bar

# ports:
#     - send: 50000
#       to: 5000
#     - send: 7777
#       to: 777
#       protocol: udp
```

4.1.2- Uso de Homestead

Para trabajar en el desarrollo del proyecto siempre tenemos que levantar homestead y además arrancar el servidor php. Cuando terminamos hay que suspender la máquina virtual para mantener el estado y parar el proceso del servidor php.

Llamadas habituales a box homestead mediante vagrant (siempre desde el directorio donde tengamos Homestead, típicamente C:\Laravel\Homestead):

- vagrant up: arranca la máquina virtual de homestead
- vagrant suspend: deja en suspensión la máquina virtual de homestead, manteniendo el estado en el que lo hayamos dejado
- vagrant reload --provision: apagamos la máquina virtual y la volvemos a encender haciendo que se ejecuten todas las provisiones. Esto hace que se eliminen los datos, por ejemplo las bases de datos.

4.1.3- Arrancar servidor mediante Artisan

Una herramienta muy potente que está incluida en Laravel es Artisan, que es unaa interfaz por línea de comandos. Utilizaremos artisan para levantar el servidor, realizar migraciones, crear modelos, etc.

Llamadas habituales a php (siempre desde el directorio donde tengamos el código, típicamente c:/Laravel/Code/necagrup):

- php artisan serve: arrancamos el servidor php desde el propio servidor para desarrollo incluido en la distribución de PHP (desde la versión 5.4). Por defecto el servidor http escuchará al puerto 8000.
- php artisan serve --port=8080 si queremos usar otro puerto.
- php artisan config:cache : en ocasiones la configuración no carga correctamente y esta llamada ha ayudado a desbloquearla en muchas ocasiones.

Con la configuración de desarrollo mencionada accederemos a la aplicación web mediante la url: http://localhost:8000/

4.2- Creación de proyecto

4.2.1- Creación de un proyecto vacío

Una vez tenemos configurado el entorno de desarrollo nos disponemos a crear un proyecto nuevo, que tendrá como nombre necagrup (ne-gocio de ca-mbios de grup-o). Para crear un proyecto nuevo utilizaremos el create-project de composer. Composer es un gestor de dependencias para php.

Ejecutamos:

- C:\Laravel\Code>composer create-project laravel/laravel necagrup --prefer-dist

Nos devuelve:

- Application key [base64:ckcYfp2XdtxvF/e6+1he5U9ptBn/yLLmYDyZ2xiI2vA=] set successfully.

4.2.2- Configuración básica del proyecto

Configuramos el archivo .env (situado en C:\Laravel\Code\necagroup). En este archivo es donde podemos encontrar la app_key (que nos devolvió la ejecución de create_project), la configuración de acceso a base de datos, de envío de emails, redis, etc.

Para nuestro caso configuramos la conexión a base de datos con los siguientes valores:

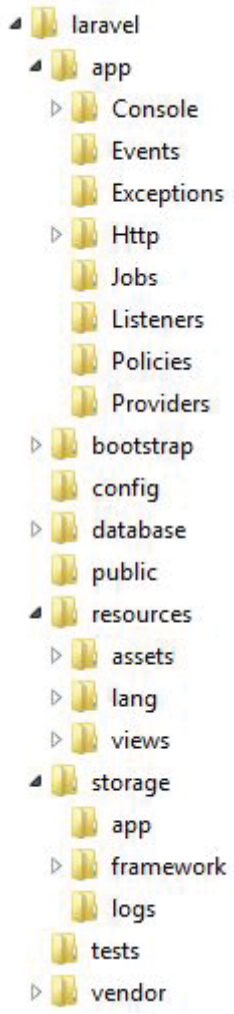
```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1:33060
DB_DATABASE=necagrup
DB_USERNAME=homestead
```

En principio la configuración usada fue
DB_HOST=localhost:3306
para que funcionase llamando a la url homestead.app o donde apunte el homestead, sin embargo, tras diversos problemas dejamos el host para que funcione en localhost:8000, dirigiendo DB_HOST a la ip de loopback y el puerto que tengamos configurado para el mysql, en este caso el 33060.
DB_HOST=127.0.0.1:33060

Comprobamos que levantando homestead y el servidor php ya tenemos acceso a localhost:8000

4.3- Ficheros de Laravel

Al crear un proyecto Laravel te genera una estructura de carpetas que es importante conocer.



Pasamos a explicar los contenidos de las carpetas más importantes:

- app: contendrá los modelos
- app/Http/Middleware: contiene el objeto necesario para la autenticación de usuarios
- app/Http/Controllers: aquí estarán los controladores
- app/Policies: políticas de autorización

- config: contiene los archivos de configuración

- database/migrations: contiene las migraciones, que son las instrucciones para generar la estructura de la base de datos que utilizará la aplicación web

- public: carpeta accesible para los usuarios, en la que guardamos, por ejemplo, las imágenes

- resources: en la carpeta resources está el frontend de la aplicación, además de los archivos de idioma si la aplicación es multilenguaje.
- resources/views: aquí estarán las vistas, siempre en un directorio con el nombre del modelo al que se refieren.

- storage: es la carpeta temporal de Laravel donde se autogeneran logs, cache de templates, etc.

- vendor: aquí estarán las dependencias y librerías del proyecto, esta carpeta la administra Composer.

4.4- Base de datos: migraciones

4.4.1- Creación de base de datos vacía

Primeramente creamos una base de datos de nombre Necagrup a través de un sistema gestor de base de datos, en este caso MySql Workbench, que será el sistema que se empleará para consultar resultados de las pruebas en base de datos.

4.4.2- Creación de tablas: definimos estructura en archivos de migración

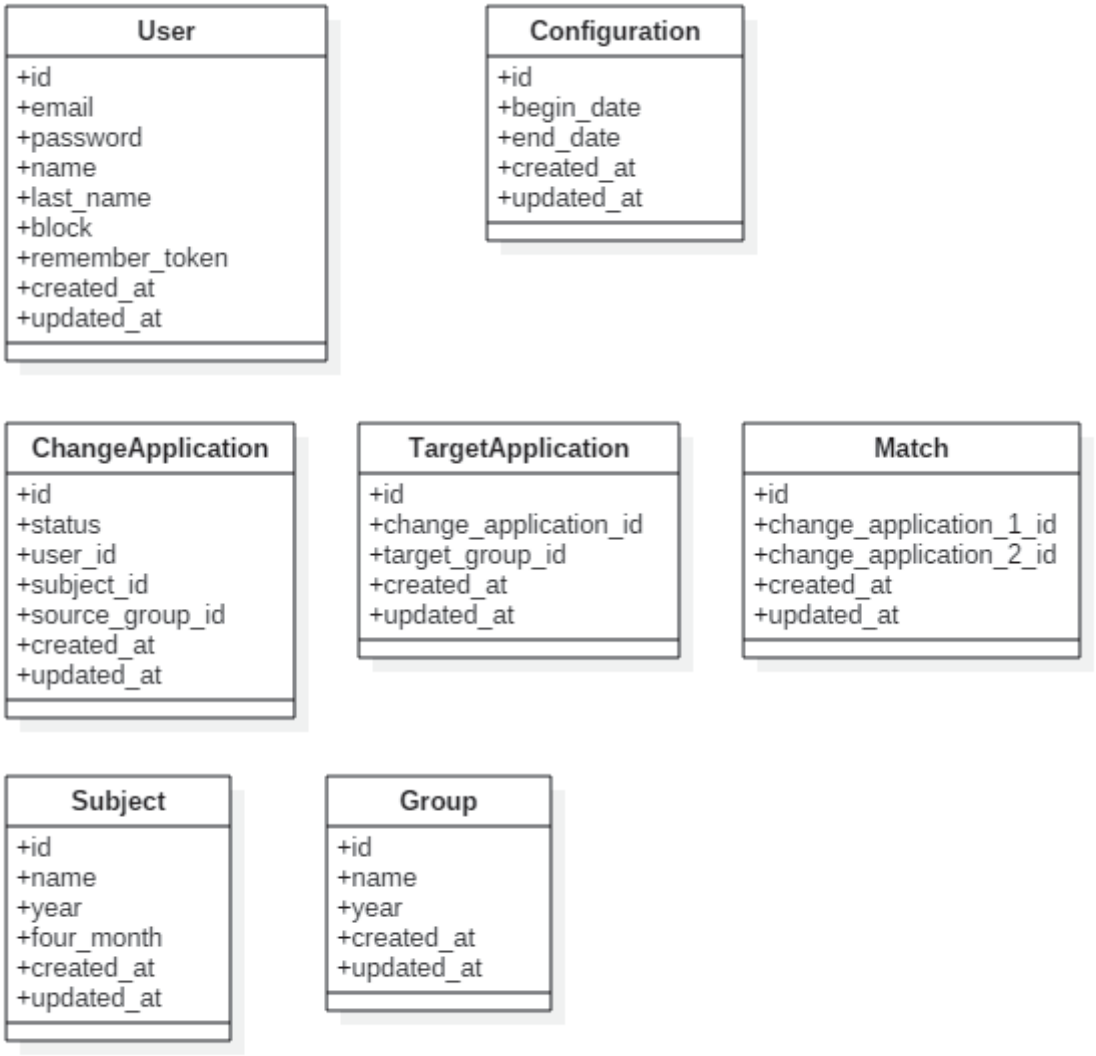
La creación de las tablas la haremos a través de migraciones, un procedimiento que tiene Laravel para hacerlo de manera cómoda y limpia.

Además de los campos contemplados hasta ahora en el diseño, al ejecutar las migraciones de Laravel aprovechamos para añadir una serie de campos que puede ser útiles en el futuro, como los timestamps created_at y updated_at, que se actualizan automáticamente al crear un registro y al modificarlo respectivamente. Estos timestamps los ponemos en todas las tablas.

También creamos un campo adicional “remember_token” en la tabla Users, ya que posibilita que el usuario pueda modificar su contraseña.

Asimismo añadimos la columna year (curso) a la tabla Group de grupos por la flexibilidad que nos da a la hora de crear el interfaz de selección de cursos, asignaturas y grupos. Ya que planteamos una interfaz en la que se puedan filtrar los grupos mediante el curso al que pertenecen.

La estructura con todos los campos presentes en la base de datos queda como podemos ver en la siguiente figura:



Por claridad hacemos un único archivo de migración por cada tabla, es decir, habrá ocho ficheros en el directorio necagroup/database/migrations, uno por cada tabla de las representadas en la figura anterior y una adicional para la tabla PasswordResets, necesaria para el funcionamiento de la autenticación.

Ejemplo de archivo de migración '2014_10_12_000000_create_users_table.php' donde creamos la tabla users e insertamos el usuario admin:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
```

```
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('email')->unique();
            $table->string('name');
            $table->string('last_name');
            $table->string('password', 60);
            $table->integer('block')->unsigned()->index();
            $table->rememberToken();
            $table->timestamps();
        });

        DB::table('users')->insert([
            'name' => 'admin',
            'email' => 'rmozun@gmail.com',
                                                                    'password'      =>
'$2y$10$8WHQEdgwdV.9.DoQ/xNDFO/VTaL.M7h9DDFK3Yhh0dsBH0GBfpg42'
        ]);
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('users');
    }
}
```

Ejemplo de archivo de migración ‘2014_10_12_100000_create_subjects_table.php’ donde creamos la tabla subjects e insertamos varias asignaturas con sus datos de año y cuatrimestre:

```
<?php
```

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateSubjectsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('subjects', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->integer('year');
            $table->string('four_month');

            $table->timestamps();
        });

                DB::table('subjects')->insert(['name' => 'Matemática Discreta', 'year' => '1',
'four_month' => '1']);
        DB::table('subjects')->insert(['name' => 'Lógica Formal', 'year' => '1', 'four_month' =>
'2']);
         DB::table('subjects')->insert(['name' => 'Fundamentos de la programación', 'year' =>
'1', 'four_month' => '1']);
          DB::table('subjects')->insert(['name' => 'Desarrollo sistemático de programas', 'year'
=> '2', 'four_month' => '1']);
        DB::table('subjects')->insert(['name' => 'Cálculo numérico', 'year' => '3', 'four_month'
=> '1']);
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('subjects');
    }
}
```

4.4.3- Correspondencia entre modelos en inglés y castellano

La implementación se ha realizado en inglés, la correspondencia entre los modelos/tablas de base de datos en inglés y los modelos planteados en el diseño, tal y como se detalló en el punto 4.1- Modelado estático: diagrama de clases y estructura BBDD, son los siguientes:

Modelo en diseño	**Modelo en implementación**
Usuario	User
Solicitud de cambio	ChangeApplication
Asignatura	Subject
Grupo	Group
Emparejamiento	Match
SolicitudDestino	TargetApplication
Configuración	Configuration

4.4.4- Población de tablas con datos de prueba

Asimismo aprovechamos las migraciones para poblar las tablas con datos por defecto para las pruebas, la creación del usuario administrador, cinco asignaturas y nueve grupos. Con estos datos hemos podido efectuar un elevado número de pruebas.

4.4.5- Creación de tablas en base de datos

Las llamadas habituales a php para ejecutar las migraciones (siempre desde el directorio donde tengamos el código):

- php artisan migrate: ejecuta las migraciones disponibles en el directorio /database/migrations que no se hayan efectuado.
- php artisan migrate:reset

4.4.6- Dificultades encontradas y miscelánea

Uno de los errores típicos al ejecutar migraciones de Laravel es olvidar que las claves foráneas tienen que establecerse como unsigned(), si no es así la migración falla y devuelve el error “Cannot add foreign key constraint”. (http://laraveldaily.com/foreign-keys-with-migrations-dont-forget-unsigned/)

La opción de resetear migraciones “php artisan migrate:reset” en ocasiones no funciona como debería, cuando se ha dado el caso hemos eliminado las tablas directamente desde el mysql workbench, hemos ejecutado “composer update” para actualizar dependencias y volvemos a hacer una migración con “php artisan migrate”.

Una vez ya tenemos toda la estructura en base de datos pasamos a crear los modelos de cada tabla.

4.5- Creación de modelos

4.5.1- Generación de modelos mediante Artisan

Para crear los modelos utilizamos las llamadas a make model que tiene disponibles Artisan.

No nos hace falta crear el modelo user, ya que está creado por defecto, así que hacemos seis peticiones a artisan para crear el resto mediante las llamadas siguientes:

```
php artisan make:model Group
php artisan make:model Subject
php artisan make:model Match
php artisan make:model Configuration
php artisan make:model ChangeApplication
php artisan make:model TargetApplication
```

4.5.2- Asignación de propiedades a atributos de modelos

Asignamos a cada modelos los atributos que serán fillable (se pueden consultar y modificar, por ejemplo el nombre o el email de un usuario), los que serán hidden (serán ocultos y no aparecerán al cargar una instancia de un modelo, por ejemplo el password de un usuario) y los que pasarán un casting a un tipo nativo al *setearse* (típicamente las claves foráneas como user_id).

4.5.3- Establecimiento de constantes para estados de modelos

Asimismo establecemos las constantes que serán necesarias para el funcionamiento del sistema, por ejemplo el modelo ChangeApplication contendrá 4 posibles valores para su campo status y 5 posibles valores de los diferentes emails que puede generar según la operación que haya realizado.

```
const STATUS_PENDING   = 'pendiente';
const STATUS_MATCHED   = 'emparejada';
const STATUS_VALIDATED = 'validada';
const STATUS_CLOSED    = 'cerrada';

const MAIL_NEW      = 'new';
```

```
const MAIL_MATCHED   = 'matched';
const MAIL_CANCELLED = 'cancelled';
const MAIL_VALIDATED = 'validated';
const MAIL_CLOSED    = 'closed';
```

4.5.4- Otros

Como el plural de match es irregular creo una excepción en las reglas de Doctrine\Common\Inflector\Inflector para que el plural sea “matches”

4.6- Autenticación y autorización

4.6.1- Creación automática de autenticación

Uno de las ventajas de usar el framework de Laravel es que dispone de una llamada por línea de comandos que directamente te crea todo lo necesario para gestionar una aplicación web que, como en este caso, necesita de autenticación.

Ejecutamos:

```
php artisan make:auth
php artisan migrate
```

Con estas llamadas ya tenemos acceso a las funcionalidades básicas de login, logout, register o reseteo de contraseña mediante el envío de un email.

4.6.2- Uso de autenticación en nuevos controladores

A partir de ahora, para que la autenticación tenga efecto en el resto de páginas de la aplicación basta con añadir en los controladores el middleware auth, por ejemplo para el controlador de solicitudes de cambio ChangeApplicationController:

```
/**
    * Create a new controller instance.
    *
    * @param  ChangeApplicationRepository  $changeapplications
    * @return void
    */
   public function __construct(ChangeApplicationRepository $changeapplications)
   {
      $this->middleware('auth');
      $this->changeapplications = $changeapplications;
   }
```

A partir de ahora para consultar al usuario autenticado empleamos la fachada Auth de la siguiente manera:

```
use Illuminate\Support\Facades\Auth;
```

```
// Get the currently authenticated user...
$user = Auth::user();

// Get the currently authenticated user's ID...
$id = Auth::id();
```

También añadimos la ruta de las autenticaciones al fichero de rutas:

```
Route::auth();
```

4.6.3- Usuario administrador

Por simplicidad asignamos al usuario administrador el id 1 de la tabla de usuarios, en el caso de esta aplicación no será necesaria una política de autorizaciones muy compleja y sólo habrá dos tipos de usuarios (un usuario administrador y el resto), por lo que no es necesaria la creación de roles de usuario.

4.6.4- Creación de política de autorización

Creo una política de autorización de las solicitudes de cambio de grupo, me servirá para restringir, por ejemplo, que un usuario sólo pueda eliminar sus propias solicitudes. Esto crea el archivo app/Policies/ChangeApplicationPolicy.php al ejecutar por línea de comandos:

```
php artisan make:policy ChangeApplicationPolicy
```

Más información: https://laravel.com/docs/5.2/quickstart-intermediate#authorization

4.6.5- Dudas de seguridad

Una duda de seguridad típica en Laravel es la ausencia de un campo salt o similar en la tabla de usuarios, lo que haría pensar en una ausencia de salteado y por lo tanto una deficiencia grave de seguridad, sin embargo en Laravel el password está salteado internamente y por ello no es necesario este campo. (https://mnshankar.wordpress.com/2014/03/29/laravel-hash-make-explained/)

En criptografía, la sal comprende bits aleatorios que se usan como una de las entradas en una función derivadora de claves. La otra entrada es habitualmente una contraseña. La salida de la función derivadora de claves se almacena como la versión cifrada de la contraseña. La sal también puede usarse como parte de una clave en un cifrado u otro algoritmo criptográfico.

Los datos con sal complican los ataques de diccionario que cifran cada una de las entradas del mismo: cada bit de sal duplica la cantidad de almacenamiento y computación requeridas.

4.6.6- Modificación de configuración de autenticación

Modificamos el controlador de autenticación, situado en app/Http/Controllers/Auth/AuthController.php para que valide los datos de un usuario al ser creado y para mapear los datos al crear una instancia de usuario.

Las validaciones por dato serán las siguientes:

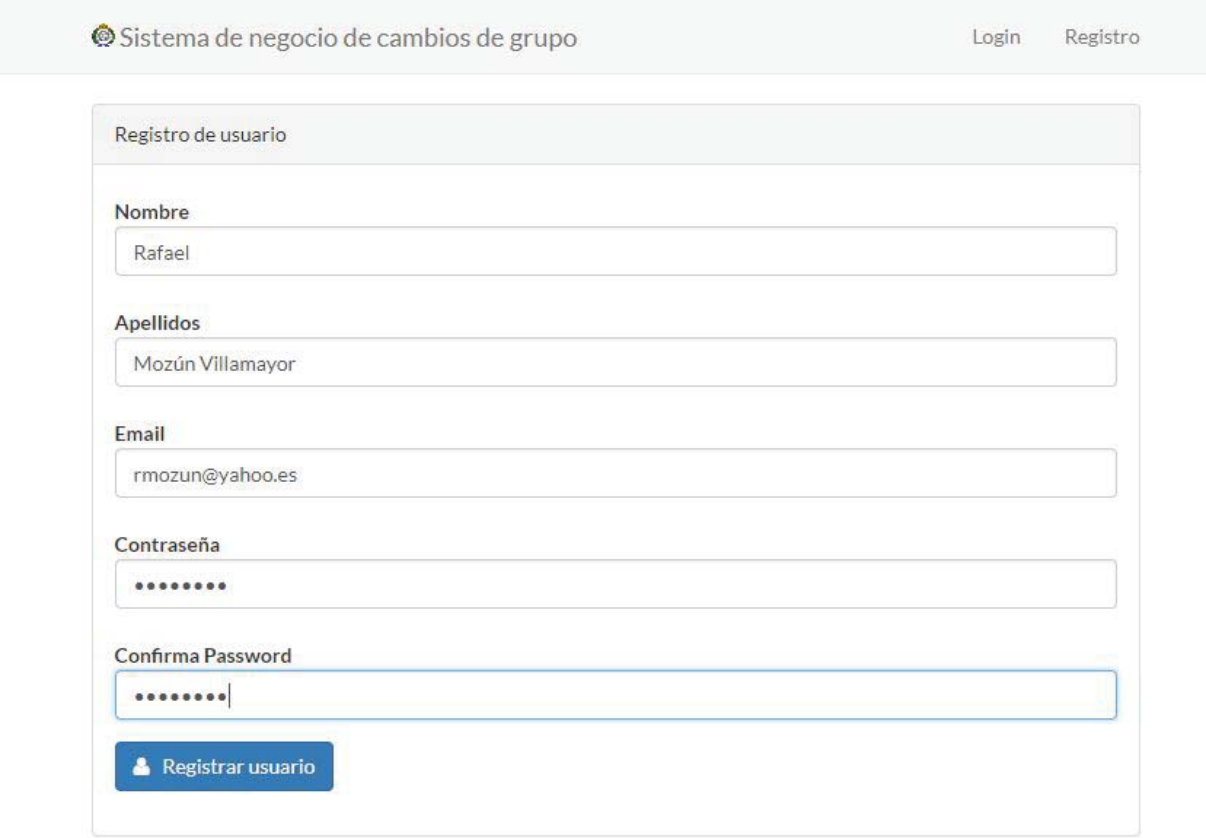
dato	**obligatorio**	**nº de caracteres**	**otras validaciones**
name	sí	máximo 255	
last_name	sí	máximo 255	
email	sí	máximo 255	único: el valor será único para todas las instancias de usuario
password	sí	mínimo 6	confirmado: deberá ser repetido en dos cajas

4.6.7- Modificación de vistas por defecto

Modificamos las vistas de registro y login para que contengan los datos que nos interesan, las figuras que se presentan a continuación ya tienen el diseño que más tarde se detallará en el punto 4.9 Interfaz gráfica de usuario.

4.6.8- Vistas definitivas

Vista definitiva de página de registro:



Vista definitiva de página de login:


Sistema de negocio de cambios de grupo
Login
Registro
Login
Email
rmozun@yahoo.es
Contraseña
••••••
Recuérdame
Entrar
He olvidado mi contraseña

4.7- Depuración y errores

A continuación una sección dedicada a depuración y errores, explicando las diferentes maneras de trazar la ejecución del código, así como la configuración de aspectos generales de la aplicación como idioma, página genérica de errores de validación, etc.

4.7.1- Depuración por log

La manera más habitual de proceder a la depuración ha sido mediante el log de Laravel que se encuentra en el path: storage\logs\laravel.log

Para facilitar su uso instalamos en la máquina de desarrollo Windows Server 2003 Resource Kit Tools, que incluye la herramienta tail para poder ver los logs más cómodamente:
(https://www.microsoft.com/en-us/download/confirmation.aspx?id=17657)

La llamada habitual para consultar el log ha sido:

```
tail -f c:\Laravel\Code\necagrup\storage\logs\laravel.log
```

4.7.2- Depuración por pantalla

Para depurar código por pantalla, esto es, ejecutando la aplicación web, simplemente añadimos en el proceso de implementación llamadas a var_dump(variable), función php que te muestra el contenido y tipo de una variable o array de forma amigable.

4.7.3- Mostrar errores de validación a usuarios

Modificamos la configuración para que los errores de validación salgan en idioma castellano. Lo hacemos en el archivo validation.php
(http://stackoverflow.com/questions/17047116/laravel-validation-attributes-nice-names)

Añadimos el archivo errors.blade.php en la ruta \resources\views\common\errors.blade.php como plantilla para que salgan los errores de validación a los usuarios.

4.7.4- Errores habituales encontrados

En el proceso de implementación han salido varios errores que, o no lo son, o forman parte del framework y hubo que parchearlos.

Error “Invalid request (Unexpected EOF)” que sale por consola, lo ignoramos según información consultada. (http://stackoverflow.com/questions/29141240/php-local-server-invalid-request-unexpected-eof)

También salta de forma, en principio aleatoria, el error “Maximum function nesting level of '100' reached, aborting!”, que lo parcheamos añadiendo en el archivo bootstrap/autoload.php la fila:

```
ini_set('xdebug.max_nesting_level', 120);
```

(http://stackoverflow.com/questions/35253984/how-to-fix-laravel-5-2-this-error-maximum-function-nesting-level-of-100-reach)

4.8- Relaciones entre modelos

4.8.1- Eloquent ORM

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional.

Laravel incluye Eloquent ORM, que te proporciona una implementación sencilla y funcional de ActiveRecord.

En Ingeniería de software, active record es un patrón de arquitectura encontrado en aplicaciones que almacenan sus datos en Bases de datos relacionales. Fue llamado así por Martin Fowler en su libro Patterns of Enterprise Application Architecture.1 La interfaz de un cierto objeto debe incluir funciones como por ejemplo insertar (INSERT), actualizar (UPDATE), eliminar (DELETE) y propiedades que correspondan de cierta manera directamente a las columnas de la base de datos asociada.

Active record es un enfoque para acceso de datos en una base de datos. Una tabla de la base de datos o vista (view) está envuelta en una clase. Por lo tanto, una instancia de un objeto está ligada a un único registro (tupla) en la tabla. Después de crear y grabar un objeto, un nuevo registro es adicionado a la tabla. Cualquier objeto cargado obtiene su información a partir de la base de datos. Cuando un objeto es actualizado, un registro correspondiente en la tabla también es actualizado. Una clase de envoltura implementa los métodos de acceso (setter y getter) o propiedades para cada columna en la tabla o vista.

Este patrón suele ser utilizado por herramientas de persistencia de objetos en el mapeo objeto-relacional. Generalmente las relaciones de llave foránea serán expuestas como una instancia de objeto de tipo apropiado por medio de una propiedad.

En eloquent ORM cada tabla de base de datos tiene su correspondiente modelo en el código, que se usará para interactuar con los datos de la tabla. Los modelos te permiten ejecutar consultas sobre los datos de la tabla, así como insertar nuevos registros, eliminarlos, etc.

Eloquent ORM es muy potente y, una vez que hemos configurado la conexión a base de datos (en config/database.php) y creados los modelos, como hemos visto en la sección 4.5, podemos dar una serie de pasos más para configurar los modelos y que nos ayuden a implementar el aplicativo de la manera más sencilla y eficiente posible.

4.8.2- Creamos relaciones entre modelos

Para seguir completando la parte de modelos empleamos la tecnología eloquent, presente en Laravel, para definir las relaciones entre los diferentes modelos y que se cumplan las restricciones del modelo del dominio.

Los tipos de relaciones soportadas por Laravel Eloquent son:

- One To One
- One To Many
- Many To Many
- Has Many Through
- Polymorphic Relations
- Many To Many Polymorphic Relations

En los siguientes subpuntos crearemos las relaciones entre modelos.

4.8.2.1- Relaciones del modelo User

Un usuario puede crear varias peticiones de cambio:

- User hasMany ChangeApplications

4.8.2.2- Relaciones del modelo ChangeApplication

Una petición de cambio pertenece a un usuario, trata sobre una asignatura, tiene un grupo origen, puede tener varios grupos de destino (a través de otro modelo/tabla) y además puede tener varios emparejamientos:

- ChangeApplication belongsTo User
- ChangeApplication belongsTo Subject
- ChangeApplication belongsTo Group (source_group)
- ChangeApplication hasManyThrough Group (target_groups)
- ChangeApplication hasMany TargetApplications
- ChangeApplication belongsTo Match

4.8.2.3- Relaciones del modelo Group

Un grupo puede pertenecer a varias peticiones de cambio, de forma directa cuando sea el grupo origen de la petición, y además puede pertenecer a varias peticiones de cambio, de manera indirecta a través del modelo targetapplications, siendo en estos casos los grupos destino escogidos en la petición de cambio:

- Group hasMany ChangeApplications
- Group hasMany TargetApplications

4.8.2.4- Relaciones del modelo Subject

Un grupo puede pertenecer a varias peticiones de cambio:

- Subject hasMany ChangeApplications

4.8.2.5- Relaciones del modelo TargetApplication

Una solicitud de destino pertenece a un grupo y a una petición de cambio:

- TargetApplication belongsTo Group
- TargetApplication belongsTo ChangeApplication

4.8.2.6- Relaciones del modelo Match

Un emparejamiento puede pertenecer a varias solicitudes de cambio, de hecho un emparejamiento pertenecerá a dos solicitudes de cambio, ambas compatibles para poder realizarse el cambio:

- Match hasMany ChangeApplications

4.8.2.7- Relaciones del modelo Configuration

El modelo Configuration no tiene ninguna relación con el resto de modelos.

4.8.3- Otras configuraciones de los modelos

Para terminar, completamos los modelos con otras configuraciones, datos asignables, casting de datos, colecciones, datos ocultos, constantes de literales, etc.

Atributos fillable: Son los atributos del modelo asignables en masa (mass asignable), es decir, que podemos asignarles valores mientras asignamos a la vez valores a otros. Si no fuera así habría que modificarlo uno a uno.

Atributos hidden: Son los atributos del modelo que no salen al mostrar la información de una instancia del modelo, típicamente son los datos relacionados con la autenticación.

Atributos casts: Algunos atributos deben realizar un casting a un tipo concreto antes de que su valor sea asignado a una instancia de un modelo, estos castings se configuran por modelo para que así se hagan de forma transparente al resto del código.

4.8.3.1- Configuración del modelo User

Asignamos al modelo User como atributos fillable los que utilizaremos para crear un nuevo usuario, y así poderlo crear directamente en un único paso. Los atributos son name, last_name, email y password.

Los atributos ocultos serán password y remember_token.

Añadimos las siguientes líneas para que la autenticación sea a través de este modelo:

```
use Illuminate\Foundation\Auth\User as Authenticatable;
class User extends Authenticatable
```

4.8.3.2- Configuración del modelo ChangeApplication

La solicitud de cambio es el modelo central del aplicativo, así que este modelo contiene mucha lógica y otros aspectos de funcionalidad de los requisitos que sólo están en este modelo. Esta lógica se irá añadiendo poco a poco, como veremos en los siguientes capítulos. En un principio configuramos lo siguiente:

Añadimos dependencia para poder escribir en log y así depurar el código:

```
use Illuminate\Support\Facades\Log;
```

Asignamos al modelo ChangeApplication como atributos fillable los que utilizaremos para crear una nueva solicitud de cambio, y así poderlo crear directamente en un único paso. Los atributos son year, status, user_id, subject_id y source_group_id.

Además predefinimos los castings que se deberán hacer a los atributos user_id y status (int y string respectivamente).

4.8.3.3- Configuración del modelo TargetApplication

Asignamos al modelo TargetApplication como atributos fillable los que utilizaremos para crear una nueva solicitud de cambio:'change_application_id y target_group_id.

Además los atributos que deben ser sujetos de casting son esos dos mismos, change_application_id y target_group_id, que serán convertidos a tipo int.

4.8.3.4- Configuración del modelo Match

El modelo de emparejamiento tendrá como atributos fillable los necesarios para crear directamente un emparejamiento, es decir: change_application_1_id y change_application_2_id.

4.8.4- Crítica

A pesar de los centenares de páginas que elogian eloquent ORM para laravel, en realidad hay casos para los que dan muchos problemas y son bastantes tutoriales los que te recomiendan que realices consultas en sql directamente, saltándote toda la abstracción de active record.

A la hora de buscar información sobre colecciones en stackoverflow también es muy habitual la crítica.

4.9- Interfaz gráfica de usuario

La interfaz gráfica de usuario o GUI (graphical user interface) es la parte de la aplicación que interactúa con el usuario, presentándole la información y las diferentes acciones que puede efectuar sobre las funcionalidades disponibles. En los últimos años se ha revalorizado este punto y suele denominarse UX (experiencia de usuario). Asimismo, a la hora de dividir un aplicativo se diferencia entre front-end y back-end, siendo la primera lo que puede ver un usuario y la segunda lo que queda invisible a él.

4.9.1- Blade

Blade es el motor de plantillas que te proporciona Lavarel, todas las vistas de Blade se compilan en código php plano y luego se cachean hasta que se modifican de nuevo. De esta manera Blade no añade ningún tipo de tiempo adicional a la carga de la web. Los archivos de vistas de blade utilizan la extensión .blade.php y están típicamente situados en el directorio /resources/views

4.9.2- Títulos, iconos y colores

Utilizaremos la librería de componentes para front-end Bootstrap. Bootstrap es open source y posiblemente la más utilizada en toda la web. Con ella es relativamente sencillo construir proyectos potentes desarrollados con HTML, CSS y JS. Es responsive y tiene una gran cantidad de componentes predefinidos, además de muchos plugins en jQuery.

Para el diseño de la aplicación emplearemos estilos CSS básicos de bootstrap 7. Asimismo, añadimos el título “Sistema de negocio de cambios de grupo”, el icono de la facultad en el path public/favicon.ico y elegimos colores neutros en grises.

4.9.3- Plantilla principal

El layout principal se situará en el archivo resources/views/layouts/app.blade.php, incluye las fuentes (librería font-awesome.min.css), los estilos (librería bootstrap.min.css), las librerías de javascript jquery.min.js y bootstrap.min.js.

A la hora de definir la plantilla o layout principal creamos una barra superior con el icono de la facultad y el nombre del aplicativo. En la parte derecha de la barra añadiremos las posibilidades de login y registro si el usuario no está logado, y si ya está logado mostraremos un pequeño menú con acceso a todas las páginas de la web: Crear solicitud de cambio, Ver cambios disponibles, Ver mis cambios y logout.

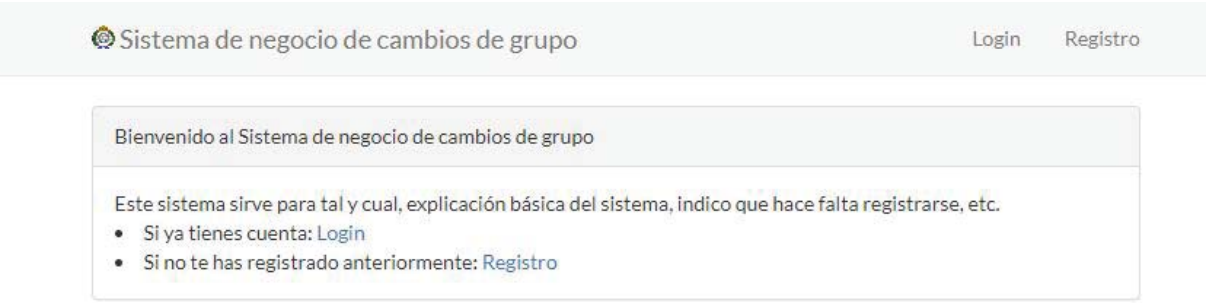
Toda la aplicación compartirá esta plantilla con la barra superior, siendo el resto de la página el contenido específico que hayamos solicitado, especificado como @yield('content') dentro del layout. Esto es una directiva de Blade que especifica que las páginas hijo pueden inyectar su propio contenido.

4.9.4- Vista de inicio

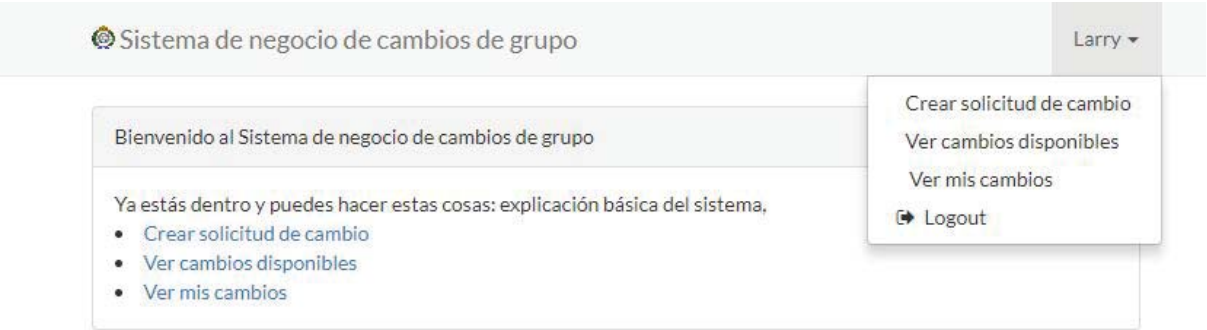
Una vez que tenemos el layout nos resta definir las vistas hijas que mostrarán el contenido principal de cada página.

Primeramente, definimos una página de inicio situada en resources/views/welcome.blade.php. Esta página, al igual que la barra de la plantilla principal, mostrará un contenido diferente dependiendo de si el usuario está logado o no. Básicamente listará las opciones posibles para cada caso, como hacíamos en la barra, pero en esta ocasión explicando mejor cada operación posible.

Ejemplo de página de inicio con barra de layout cuando no hay ningún usuario logado:



Ejemplo de página de inicio con barra de layout cuando el usuario está logado:



4.9.5- Vista de errores

Una vista importante es la de errores, añadimos el archivo errors.blade.php en la ruta \resources\views\common\errors.blade.php como plantilla para que salgan, por ejemplo, los errores de validación de los usuarios.

Simplemente alerta de la existencia de algún error “Vaya, parece que algo no va bien” y lista todos los errores que haya en la variable $errors. Esta variable está disponible para todas las vistas de Laravel, siendo una instancia de ViewErrorbag vacía si no hay errores de validación.

De esta forma podremos mostrar los posibles errores de validación en el mismo formato da igual en qué página se haya producido.

A partir de ahora incluiremos el listado de errores en las páginas que queramos con el siguiente código:

```
<!-- Display Validation Errors -->
@include('common.errors')
```

4.9.6- Cambios en traducciones

En este punto revisamos los textos de validación y en el archivo resources\lang\es\validation.php hacemos un cambio, introduciendo el texto "El campo email ya existe en el sistema." que sustituye a "El campo email ya ha sido tomado."

Aprovecho para establecer el locale de la aplicación a “es” en config/app.php y cambiar los errores de validación para que salgan en castellano.

Cambio nombre de attributes en castellano en validation.php según:
http://stackoverflow.com/questions/17047116/laravel-validation-attributes-nice-names

4.9.7- Cambios en rutas

También hacemos una serie de cambios en el archivo de rutas:

Establecemos la vista welcome como la vista por defecto cuando el usuario quiera acceder al directorio raíz (/) de la web:

```
Route::get('/', function () {
    return view('welcome');
```

```
});
```

4.9.8- Vistas de autenticación

Las vistas de autenticación ya vienen predefinidas en el directorio resources/views/auth, simplemente tenemos que traducir los textos y cambiar las páginas de login y registro (login.blade.php y register.blade.php respectivamente) para que contenga los campos que necesitamos. Así que añadimos el campo last_name/apellidos, lo ponemos como obligatorio en el controlador de autenticación situado en app\Http\Controllers\Auth\AuthController.php

Duplico para /resources/lang/es los archivos de /resources/lang/en y los copio de: https://github.com/MarcoGomesr/laravel-validation-en-espanol , excepto auth, que lo traduzco yo y algunas cosas que cambian en validation. Traduzco también email.blade.app y reset.blade.php

4.9.9- Problemas en uso de forms

Tal y como se ha indicado en puntos anteriores, hay algunos conflictos entre las diferentes versiones de Laravel. En este momento intentamos crear componentes para las vistas y algunos, como los desplegables, dan muchos problemas.

Investigamos y detectamos que es un fallo habitual, con no poca polémica en foros, en todo caso modificamos algunas partes de la aplicación para poder usar helpers de forms, que nos facilitan mucho la vida sin tener que escribir directamente código html.

El error que salía era “class 'Form' not found” y tal como se indica en http://stackoverflow.com/questions/28753767/laravel-5-class-form-not-found o incluso en la web oficial de Laravel: https://laravel.com/docs/5.1/upgrade#upgrade-5.0 hacemos una serie de modificaciones para que nos funcione.

El problema parte de que los helpers han sido deprecados para Laravel 5, y ha sido la comunidad de desarrolladores de este lenguaje los que han creado un reemplazo.

Utilizaremos LaravelCollective (https://laravelcollective.com/docs/5.1/html), añadiendo esta dependencia en nuestro archivo composer.json:

```
"require": {
        "laravelcollective/html": "~5.0"
}
```

También sumamos una línea en confif/app.php para el provider:

```
'Collective\Html\HtmlServiceProvider',
```

E incluimos un par de aliases en config/app.php para las fachadas:

```
'Form' => 'Collective\Html\FormFacade',
'Html' => 'Collective\Html\HtmlFacade',
```

4.10- Crear solicitud de cambio

La primera operación de los casos de uso es la creación de una solicitud de cambio. A continuación explicamos los cambios realizados sobre cada parte del aplicativo para implementar dicha funcionalidad.

4.10.1- Archivo rutas

Añadimos una llamada ajax para cargar el listado de grupos existentes en un año del curso, o el listado de asignaturas para un año y cuatrimestre concretos. Se usará para cargar datos dinámicamente en los desplegables.

También añadimos al fichero de rutas las correspondientes a la creación de una solicitud de cambio:

```
Route::get('/changeapplication', 'ChangeApplicationController@create');
Route::post('/changeapplication', 'ChangeApplicationController@store');
```

4.10.2- Funciones javascript en layout principal

Creamos una serie de funciones javascript en el layout principal de la aplicación para que carguen los desplegables de Asignaturas y Grupos al seleccionar o cambiar el desplegable del Año/curso académico, o sólamente el de Asignaturas al seleccionar o cambiar el desplegable del cuatrimestre.

En estos casos se cargarán en los desplegables, de forma dinámica y sin necesidad de recargar toda la página, los datos con las asignaturas y grupos que cumplen los filtros deseados.

4.10.3- Cambios en modelo User

Creamos la función isadmin en modelo user que te devuelve si el usuario es el administrador.

4.10.4- Cambios en modelo ChangeApplication

Añadimos constantes para almacenar literales que luego estarán en base de datos. Las constantes serán para el estado de la solicitud de cambio:

```
const STATUS_PENDING   = 'pendiente';
const STATUS_MATCHED   = 'emparejada';
const STATUS_VALIDATED = 'validada';
const STATUS_CLOSED    = 'cerrada';
```

4.10.5- Cambios en el controlador de ChangeApplication

Creamos en el controlador de ChangeApplication las dos acciones que hemos reflejado en el archivo de rutas.

La acción create devuelve la vista changeapplications.new

La acción store crea la solicitud de cambio recibida como parámetro, previamente validada, ya que son campos requeridos el año, el cuatrimestre, la asignatura, el grupo origen y al menos un grupo de destino.
Para el caso del usuario administrador añadimos que la solicitud de cambio no siempre sea asignada al usuario que la ha creado, ya que el administrador dispondrá de un desplegable con todos los alumnos para poder crear una solicitud en su nombre.

Una vez validada correctamente la solicitud de cambio creamos el modelo correspondiente, con estado por defecto a “pendiente”. Al crear una instancia de este modelo también se crean una o varias del modelo TargetGroup.

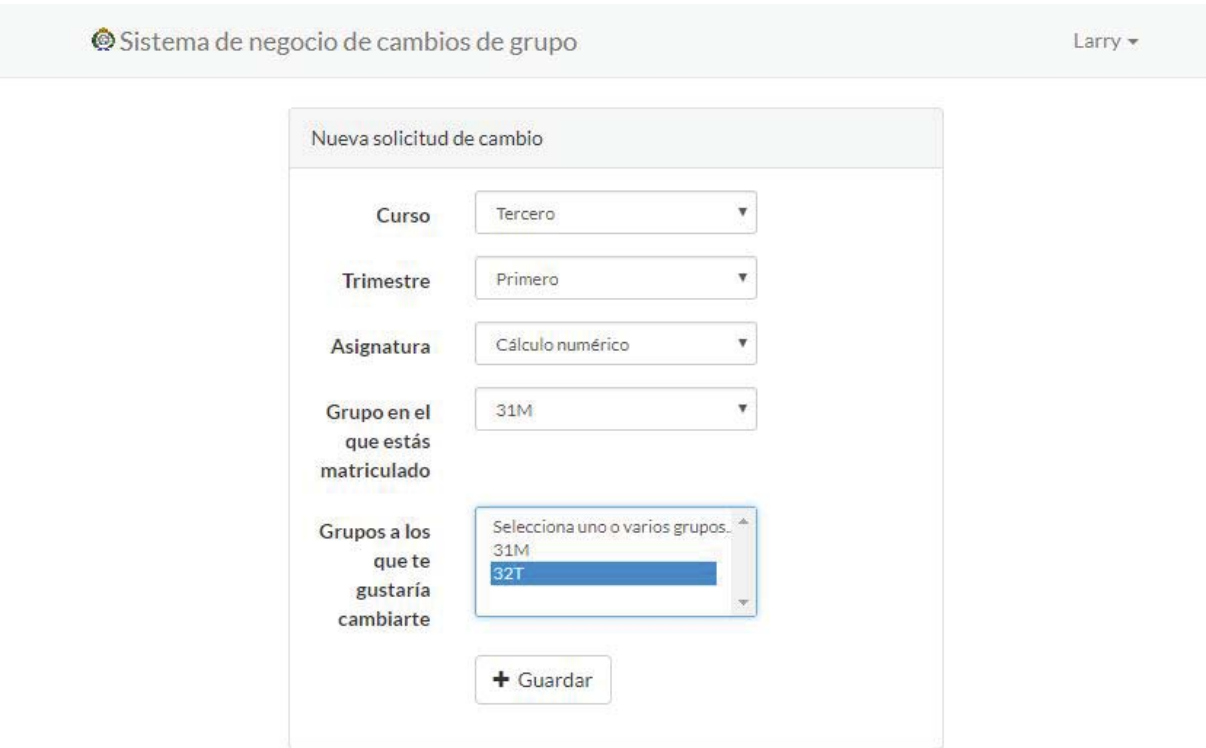
Para finalizar enviamos un email al usuario notificándole su acción (ver sección de emails), ejecutamos la acción que busca emparejamientos para la solicitud recién creada y redirigimos a la página de Ver solicitudes de cambio.

4.10.6- Vistas ChangeApplication

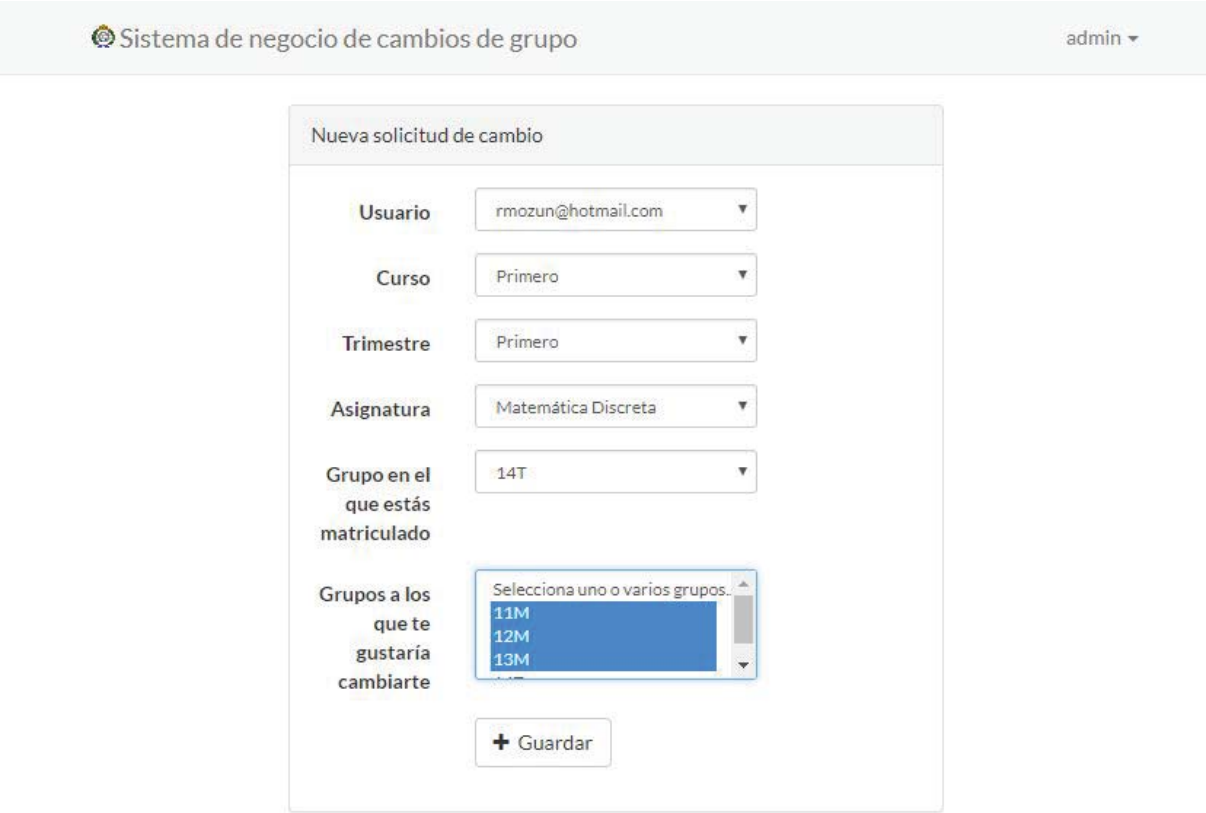
Creamos vista para nueva solicitud de cambio con desplegables para seleccionar los diferentes campos que deben contener una solicitud.

Ejemplo de página de Nueva solicitud de cambio:

Ejemplo de página de Nueva solicitud de cambio con datos introducidos:



Ejemplo de página de Nueva solicitud para administrador:



4.11- Ver solicitudes de cambio

La siguiente operación de los casos de uso es el listado de solicitudes de cambio disponibles y las posibles operaciones sobre ellas. A continuación explicamos los cambios realizados sobre cada parte del aplicativo para implementar dicha funcionalidad.

4.11.1- Archivo rutas

Añadimos al fichero de rutas las correspondientes al listado (index) de las solicitudes del usuario, así como dos rutas para el listado de solicitudes disponibles:

```
Route::get('/changeapplications', 'ChangeApplicationController@index');

Route::get('/available', 'ChangeApplicationController@available');
Route::post('/available', 'ChangeApplicationController@available');
```

4.11.2- Cambios en el controlador de ChangeApplication

La acción index del controlador de solicitudes de cambio devuelve la vista index de changeapplication son la variable changeapplications cargada con todas las solicitudes de cambio pertenecientes al usuario logado.

La acción available devuelve la vista available, es decir, las solicitudes de cambio disponibles, filtradas por asignatura si ésta se ha introducido a través del interfaz.

Para obtener las solicitudes disponibles para un usuario y una asignatura empleamos un repositorio. Para poderlo utilizar, en esta parte del controlador simplemente indicamos la dependencia de la manera habitual e indicamos que el constructor del controlador sea de tipo ChangeApplicationRepository

```
use App\Repositories\ChangeApplicationRepository;

public function __construct(ChangeApplicationRepository $changeapplications)
```

4.11.3- Cambios en modelo ChangeApplication

Creamos funciones para consultar el estado de una solicitud de cambio de manera sencilla, devolviendo un booleano: is_pending(), is_validated()

Establecemos que el modelo ChangeApplication sea de tipo repositorio. El patrón de repositorio fue presentado por primera vez por Eric Evans en su libro de diseño dirigido por el dominio (Domain Driven Design: Tackling Complexity in the Heart of Software). El repositorio es, de hecho, el punto de entrada para que la aplicación acceda a la capa de dominio.

Por simplificar, el repositorio permite que todo su código utilice objetos sin tener que saber cómo se persiste el objeto. El repositorio contiene todo el conocimiento de la persistencia, incluida la asignación de tablas a objetos. Esto proporciona una vista más orientada a objetos de la capa de persistencia y hace que el código de asignación sea más encapsulado.

Creamos nuevo ficheroapp/repositories/ChangeApplicationRepository.php y creamos las funciones forUser (devuelve solicitudes de cambio para un usuario concreto), availableForUser (devuelve las disponibles para ser reservadas por el usuario), y availableForUserAndSubject (que además te filtra por las de una asignatura en concreto).

4.11.4- Vistas ChangeApplication

Creamos una vista para poder ver todas las solicitudes del usuario actual, con la información de cada solicitud y botones para las acciones que se puedan realizar sobre ellas.

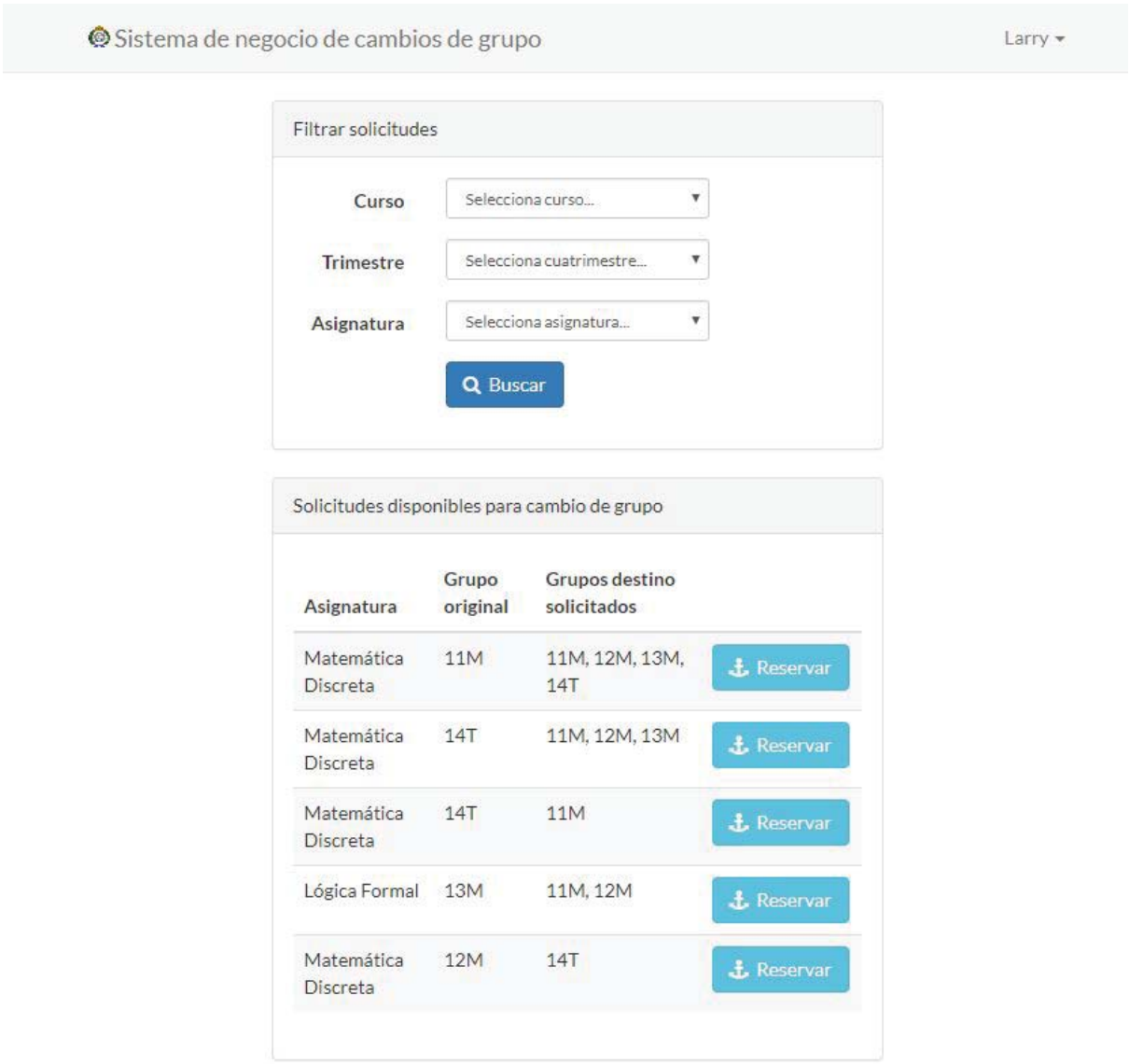
Los diferentes estados de cada solicitud vienen en diferentes colores para una mejor comprensión.

Asimismo, incluimos un recuadro de “Solicitudes en bloque” para poder consultar y modificar dicha configuración.

También creamos un recuadro para poder filtrar las solicitudes por curso, trimestre o asignatura. La acción de reservar solicitud la tratamos en el capítulo siguiente.

Ejemplo de página de Ver mis solicitudes:

Ejemplo de página de Ver solicitudes disponibles:


Sistema de negocio de cambios de grupo
Larry
Filtrar solicitudes
Curso
Selecciona curso...
Trimestre
Selecciona cuatrimestre...
Asignatura
Selecciona asignatura...
Buscar
Solicitudes disponibles para cambio de grupo
Asignatura
Grupo original
Grupos destino solicitados
Matemática Discreta
11M
11M, 12M, 13M, 14T
Reservar
Matemática Discreta
14T
11M, 12M, 13M
Reservar
Matemática Discreta
14T
11M
Reservar
Lógica Formal
13M
11M, 12M
Reservar
Matemática Discreta
12M
14T
Reservar

4.12- Reservar cambio

La siguiente operación de los casos de uso consiste en reservar un cambio de los disponibles. A continuación, explicamos los cambios realizados sobre cada parte del aplicativo para implementar dicha funcionalidad.

4.12.1- Archivo rutas

Añadimos al fichero de rutas las correspondientes a la creación y posterior salvaguarda de la reserva de una solicitud de cambio:

```
Route::get('/bookchangeapplication/{changeapplication}',
'ChangeApplicationController@createbook');

Route::post('/bookchangeapplication', 'ChangeApplicationController@storebook');
```

4.12.2- Cambios en el controlador de ChangeApplication

Añadimos acciones para crear y salvar una reserva de solicitud, en realidad lo que hacemos es crear una nueva solicitud de cambio y le asignamos los datos adicionales necesarios para que ya esté empareja con la solicitud reservada.

Es por esto que ya la nueva solicitud ya está en estado “validada” desde su creación, mientras que la reservada pasa a estado “emparejada”. Tras estas operaciones de creación se envían sendos mails a los usuarios notificándoles de la acción.

4.12.3- Cambios en el modelo ChangeApplication

Añadimos funciones que realicen operaciones sobre solicitud de cambio, típicamente cambiar de estado y enviar los emails que requiera ese cambio de estado: cancel(), validate(), close().

Las funciones match_it(), being_matched() son un poco más complejas, ya que al crearse un emparejamiento se modifican dos solicitudes de cambio.

4.13- Gestionar solicitudes de cambio

La siguiente operación de los casos de uso consiste en gestionar las propias solicitudes de cambio y las diferentes acciones que podemos realizar sobre ellas. A continuación, explicamos los cambios realizados sobre cada parte del aplicativo para implementar dicha funcionalidad.

4.13.1- Archivo rutas

Añadimos al fichero de rutas las correspondientes a las acciones de cancelar, validar o eliminar una solicitud de cambio, además de la de modificar la configuración de bloque para el usuario:

```
Route::put('/changeapplication/cancel/{changeapplication}',
'ChangeApplicationController@cancel');

Route::put('/changeapplication/validate/{changeapplication}',
'ChangeApplicationController@validatechange');

Route::put('/changeapplication/block', 'ChangeApplicationController@block');

Route::delete('/changeapplication/{changeapplication}',
'ChangeApplicationController@destroy');
```

4.13.2- Cambios en el controlador de ChangeApplication

Creamos varias acciones para cancelar una solicitud, validar una solicitud, cambiar la configuración de bloque del usuario o eliminar una solicitud:

- Cancelar una solicitud, implica, además del cambio de estado de la solicitud, cancelar otra solicitud si la primera ha sido emparejada con otra, así como eliminar dicho emparejamiento.

- Validar una solicitud de cambio consiste en cerrar la solicitud (ya que se ha procesado con éxito, existiendo otra solicitud complementaria) si la solicitud emparejada está validada (cerrando también ésta última), o bien, si la emparejada no está validada simplemente la cambio a estado validada.

- Creamos una acción para modificar la configuración de bloque del usuario.

- Eliminar una solicitud pide antes autorización para ver si el usuario puede eliminarla.

También tenemos en cuenta que una solicitud de cambio que pertenezca a un bloque sólo podrá cerrarse (pasar a estado Cerrada) si todas las solicitudes de dicho bloque están en estado Validada.

4.13.3- Cambios en el modelo ChangeApplication

Aprovechamos para crear automatización de buscar emparejamientos:

Añadimos dependencias de base de datos para poder hacer consultas complejas:

```
use DB;
```

También creamos la función search_matches(), que busca entre todas las aplicaciones de cambio en estado pendiente por alguna que pueda emparejarse con la actual. Se llama cuando la solicitud de cambio acaba de ser creada o bien ha cambiado su estado a pendiente.

Matcheo /emparejamiento es cuando encontramos otra solicitud de cambio cuyo el origen sea alguno de mis destinos y mi origen alguno de sus destinos

Las consultas son complicadas con llamadas habituales de activerecord, así que lo hacemos con DB, haciendo llamadas más parecidas a queries sql normales.

Si encontramos algún match entonces se efectúa el emparejamiento.

Creamos función block(valor) en modelo user que asigna directamente el valor de bloque al usuario objeto de la llamada.

4.13.4- Cambio en políticas de autorización

Creo función en archivo de políticas de autorización situado en: app/Policies/ChangeApplicationPolicy.php para determinar que sólo el propio usuario al que pertenece una solicitud de cambio pueda eliminarla.

4.14- Mailing

4.14.1- Configuración de envío mail

Realizamos cambios en el fichero de configuración de mail, situado en config/mail.php para realizar nuestras pruebas:

```
'driver' => env('MAIL_DRIVER', 'smtp'),
'host' => env('MAIL_HOST', 'localhost'),
'port' => env('MAIL_PORT', 25),
'encryption' => env('MAIL_ENCRYPTION', ''),
```

4.14.2- Tipos de correos enviados

Añadimos en el modelo ChangeApplication constantes para almacenar literales. Las constantes serán para los diferentes tipos de email que se puede enviar a un usuario dependiendo de la operación que se haya realizado:

```
const MAIL_NEW      = 'new';
const MAIL_MATCHED   = 'matched';
const MAIL_CANCELLED = 'cancelled';
const MAIL_VALIDATED = 'validated';
const MAIL_CLOSED    = 'closed';
```

4.14.3- Cambios en modelo ChangeApplicacion

Añadimos dependencia de email para poder enviar correos electrónicos:

```
use Mail;
```

Creamos la llamada send_email(type), para poder enviar todos los tipos de email necesarios para la aplicación. Son emails de texto plano que constan de asunto y cuerpo del mensaje. Se deberán personalizar y añadir enlaces según las necesidades futuras del sistema.

tipo email	asunto	texto
new	Nueva solicitud de cambio de grupo	Has creado una nueva solicitud de cambio de grupo con los siguientes datos...
matched	Tu solicitud de cambio de grupo ahora está emparejada	La solicitud que cursaste en fecha tal con los datos estos ahora está emparejada, puedes efectuar las siguientes acciones...
validated	Tu solicitud de cambio de grupo ahora está validada	La solicitud que cursaste en fecha tal con los datos estos ahora está validada, puedes efectuar las siguientes acciones...'
cancelled	Tu solicitud de cambio de grupo ha sido cancelada	La solicitud que cursaste en fecha tal con los datos estos has sido cancelada, puedes efectuar las siguientes acciones...
closed	Tu solicitud de cambio de grupo ahora está cerrada	'La solicitud que cursaste en fecha tal con los datos estos ahora está cerrada, puedes efectuar las siguientes acciones...

Además configuramos por defecto como sender al from siguiente:
‘cambios_de_grupo@fi.upm.es', 'Cambios de grupo FI’

4.15- Pruebas

Las pruebas y depuración del sistema han ido realizándose en cada punto de implementación del aplicativo, las formas más habituales de depuración ya fueron comentadas en el punto 4.7

4.15.1- Pruebas realizadas en cada iteración

Al haberse realizado la implementación por etapas según los casos de uso, con algunas excepciones, hemos aprovechado el término de cada iteración para realizar pruebas en profundidad y para cada posible escenario planteado en los casos de uso en formato extendido.

4.15.2- Datos de prueba

Al ejecutar la migración de las tablas en la base de datos también hemos añadido datos de prueba:

- Tabla users: añadimos usuario admin con nombre admin, id 1 y email a rmozun@gmail.com

- Tabla groups: creamos 9 registros de grupos, alternando años y turnos, con la nomenclatura habitual de la facultad: 11M, 12M, 14T, 21M, 32T, etc.

- Tabla subject: añadimos varias asignaturas, indicando el curso académico/año y cuatrimestre al que pertenecen, las asignaturas son todas pertenecientes al plan 96 por motivos nostálgicos: Matemática discreta, Lógica formal, Desarrollo sistemático de programas, Cálculo numérico, etc.

Con estos datos añadidos de forma automática ya podemos hacer pruebas en profundidad de todo el sistema.

4.15.3- Pruebas de sistema

Una vez tenemos todo el sistema implementado, hacemos pruebas de nuevo haciendo hincapié en los ciclo de uso que creemos que serán más habituales en el uso práctico del sistema, es decir:

Operación	Crear usuario	Usuario crea solicitud de cambio	Usuario reserva solicitud de cambio
Acciones	- Creo usuario - Usuario se loga con éxito	- Usuario crea solicitud de cambio - Solicitud se enlaza automáticamente a otra preexistente - Valido cambio - Otra persona valida - Cambio realizado con éxito	- Reservo cambio a partir de solicitud preexistente - Otra persona valida - Cambio realizado con éxito

4.15.4- Depuración y pruebas de mailing

Para realizar las pruebas de envío de correos hemos utilizado un servidor fake de smtp (http://smtp4dev.codeplex.com/), de esta manera podemos probar en local sin tener que depender de un servidor smtp externo.

El servidor fake arranca simplemente al ejecutar smtp4dev.exe

5- Implantación

En este punto explicaremos, a modo de manual de instalación para el usuario, lo que tiene que hacer el administrador del sistema para implantar este aplicativo en su servidor. Presuponemos que el servidor donde se va a instalar ya dispone de un servidor de base de datos MySQL, además de PHP y un servidor web tipo Nginx

5.1- Archivos de la web

Se adjuntan todos los archivos necesarios para ejecutar la web, estos se encuentran en el directorio /necagroup, que es donde tiene que apuntar la instancia del servidor web.

5.2- Base de datos, creación de tablas y precarga de datos

Crearemos desde nuestro servidor de base de datos una base de datos de nombre, por ejemplo, necagroup.

Para crear las tablas y los datos por defecto utilizaremos, tal y como hicimos en la implementación, Artisan, la interfaz por línea de comandos de Laravel.

Las llamadas habituales a php para ejecutar las migraciones (siempre desde el directorio donde tengamos el código):

- php artisan migrate: ejecuta las migraciones disponibles en el directorio /database/migrations que no se hayan efectuado.

5.3- Configuración básica de la web

Configuramos el archivo .env (situado en C:\Laravel\Code\necagroup), en este archivo es donde podemos encontrar la configuración de acceso a base de datos y de envío de emails, que son las que necesitamos para este proyecto:

Para nuestro caso configuramos la conexión a base de datos con los siguientes valores:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1:33060
```

```
DB_DATABASE=necagrup
DB_USERNAME=homestead
```

Además para la configuración de email:

```
MAIL_DRIVER=smtp
MAIL_HOST=localhost
MAIL_PORT=25
MAIL_USERNAME=rmozun@gmail.com
MAIL_PASSWORD=1122334455
MAIL_ENCRYPTION=
```

Comprobamos que levantando el servidor php ya tenemos acceso a la web, típicamente en http://localhost:8000

Ante cualquier problema consultamos el log de la aplicación:
tail -f C:\Laravel\Code\necagrup\storage\logs\laravel.log

5.4- Procedimiento de reseteo

Por claridad tenemos un único archivo de migración por cada tabla, es decir, habrá ocho ficheros en el directorio necagroup/database/migrations, uno por cada tabla de las representadas en la figura anterior y una adicional para la tabla PasswordResets, necesaria para el funcionamiento de la autenticación.

Las llamadas habituales a php para ejecutar las migraciones (siempre desde el directorio donde tengamos el código):

- php artisan migrate:reset ejecuta las migraciones desde la primera hasta la última, no sólamente las que no se hayan efectuado con anterioridad.

Nota: es muy habitual que "php artisan migrate:reset" no funcione como debe, el *walkaround* consiste en ejecutar "composer update" y en eliminar todas las tablas antes de ejecutar de nuevo "php artisan migrate"

6- Conclusiones

Hacer el Proyecto de Fin de Carrera 7 años después de aprobar la última asignatura y con casi 14 años de experiencia en el mundo laboral ha hecho que la consecución de este proyecto haya sido muy diferente de haberlo hecho cuando es habitual para otros alumnos.

La parte de diseño ha sido la más lenta y laboriosa debido a que son tareas que se omiten en la vida real de la empresa de desarrollo -con la excepción de proyectos muy grandes- teniendo la tutora que volver a explicarme muchos conceptos, algo que no hubiera sido necesario para un alumno que recientemente hubiera aprobado las asignaturas de ingeniería del software.

Respecto a la implementación, el tiempo de desarrollo total fue relativamente corto, dada mi larga experiencia en este campo, el código es sencillo de mantener, las secciones más complejas están bien documentadas en el propio código y ante posibles dudas de configuración o usabilidad -que se despejarán con el uso real- he dejado el código lo más genérico posible para flexibilizar el futuro de la aplicación.

A la hora de codificar, los mayores problemas fueron montar el entorno de desarrollo y la configuración de los mailers, así como, de forma general, resolver cada problemática que tuviera algún tipo de “polémica” en la comunidad Laravel, al encontrar muchas versiones contradictorias según la fuente investigada. Toda esta información está documentada en esta memoria y en todo momento he aportado referencias y razones por las que se opta por una solución en concreto.

Aunque el tiempo de desarrollo en horas totales no ha sido muy grande, sí que se ha dilatado mucho en el tiempo por las dificultades para compaginar este PFC con un trabajo que exige más de 50 horas semanales, además del periodo normal de adaptación a una tecnología, Laravel, desconocida para mí al comienzo de este proyecto.

Más allá de cuestiones técnicas, la mayor dificultad en la consecución de este PFC ha sido encontrar huecos de tiempo suficientes para ser productivo, tanto en número de horas consecutivas como en días dedicados al proyecto. Por simplificar diremos que no ha sido un proyecto que se haya completado dedicando 2 horas al día, sino 10 horas cada día que fue físicamente posible, prácticamente siempre en periodos vacacionales.

7- Líneas futuras

Las posibles líneas futuras de esta aplicación incluyen integraciones con otros sistemas, por ejemplo la autenticación de los usuarios mediante SSO (Single Sign-On) a través del LDAP del Centro de Cálculo de la Facultad, o la importación de los datos de asignaturas y grupos a través de alguna API que proporcione la UPM.

También se podría haber implementado el soft deleting (para poder recuperar datos borrados accidentalmente o por error), que los alumnos puedan consultar el listado de solicitudes de cambio antes de tener que registrarse, que puedan darse de baja o que el registro incluya más datos que faciliten la gestión final para secretaría, como el dni, aunque incluir más datos también requeriría de otro tipo de desarrollos (cláusulas legales, aceptar condiciones, etc) para cumplir la lopd.

Asimismo, siempre se puede perfeccionar el diseño y la usabilidad, y la mejor manera de descubrir posibles mejoras en estos puntos es detectarlas cuando la aplicación entre en producción para usuarios reales.

8- Referencias

Todas las referencias aportadas estaban online a fecha de 31 de marzo de 2018.

8.1- Arquitectura Laravel y referencia básica

https://laravel.com
https://laravel.com/docs/5.2
https://es.wikipedia.org/wiki/Laravel

Arquitectura de Laravel:
https://platzi.com/blog/arquitectura-laravel/
https://styde.net/porque-laravel-no-es-mvc-y-tu-deberias-olvidarte-de-mvc/

Laravel y la filosofía MVC
https://styde.net/porque-laravel-no-es-mvc-y-tu-deberias-olvidarte-de-mvc/

Código de versión de Laravel empleada:
https://github.com/laravel/framework/releases/tag/v5.2.45

Versión de Laravel empleada:
http://www.elcoderino.com/check-laravel-version/

8.2- Instalación de Laravel y nuevo proyecto

Tutoriales y ejemplos:
https://styde.net/instalacion-de-composer-y-laravel-en-windows/
http://tech.osteel.me/posts/2015/04/23/how-to-start-a-new-laravel5-project-with-homestead-quick-reference.html
https://github.com/laravel/quickstart-basic
https://laravel.com/docs/5.2/quickstart-intermediate

Composer:
https://getcomposer.org/download/

Manuales y tutoriales Laravel:
http://www.desarrolloweb.com/manuales/manual-laravel-5.html
https://www.packtpub.com/books/content/laravel-50-essentials

Logging:
https://www.easylaravelbook.com/blog/logging-an-array-in-laravel/

http://laraveldaily.com/echoing-dd-vs-var_dump-vs-print_r/

Windows Server 2003 Resource Kit Tools:
https://www.microsoft.com/en-us/download/confirmation.aspx?id=17657

8.3- Homestead, vagrant y entorno de desarrollo

https://laravel.com/docs/5.2/homestead
https://laracasts.com/index.php/discuss/channels/general-discussion/what-is-the-proper-way-to-refresh-homestead-after-making-changes-to-the-homesteadyaml-file?page=1
https://styde.net/instalar-laravel-homestead-en-windows/
https://scotch.io/tutorials/getting-started-with-laravel-homestead
https://desarrolloweb.com/articulos/instalar-homestead-para-laravel5.html
https://www.vagrantup.com/intro/index.html
https://styde.net/aprende-a-instalar-y-usar-vagrant-y-laravel-homestead/

Problema con VirtualBox can't find host-only adapters on Windows 10 host => Fixed in SVN
https://www.virtualbox.org/ticket/14437

8.4- Diseño y maquetación

Botones y tablas bootstrap:
https://www.w3schools.com/bootstrap/bootstrap_buttons.asp
https://www.w3schools.com/bootstrap/bootstrap_ref_css_buttons.asp
https://www.w3schools.com/bootstrap/bootstrap_tables.asp

Iconos fontawesome:
http://astronautweb.co/snippet/font-awesome/
https://www.w3schools.com/icons/fontawesome_icons_webapp.asp

Ajax en laravel:
https://stackoverflow.com/questions/38505750/laravel-5-fetch-ajax-data-in-route-and-pass-to-controller
https://laracasts.com/discuss/channels/general-discussion/laravel-5-ajax-get-and-post-examples
https://laracasts.com/discuss/channels/laravel/form-redirecting-to-route-on-ajax-request

8.5- Formularios y componentes

Filtros por url:
https://laracasts.com/discuss/channels/laravel/laravel-5-query-builder-search-filter?page=1

Confirmación al eliminar:
https://laravel-tricks.com/tricks/confirmation-before-deleting-an-item

Carga de desplegables:
https://stackoverflow.com/questions/29508297/laravel-5-how-to-populate-select-box-from-database-with-id-value-and-name-value

Problemas con forms:
https://laravelcollective.com/docs/5.2/html#drop-down-lists
http://anytch.com/creating-dynamic-select-drop-down-input-with-laravel-5/2/
https://stackoverflow.com/questions/28753767/laravel-5-class-form-not-found
http://laravel-recipes.com/recipes/124/opening-a-new-html-form
http://anytch.com/creating-dynamic-select-drop-down-input-with-laravel-5/4/

El error que salía era “class 'Form' not found” y tal como se indica en http://stackoverflow.com/questions/28753767/laravel-5-class-form-not-found o incluso en la web oficial de Laravel: https://laravel.com/docs/5.1/upgrade#upgrade-5.0

LaravelCollective:
https://laravelcollective.com/docs/5.1/html

8.6- Base de datos y consultas

Problemas con base de datos:
https://github.com/jatubio/ja_duilio_curso-laravel-5/wiki/Configurar-la-Base-de-Datos-en-Laravel-5
http://tech.osteel.me/posts/2015/04/23/how-to-start-a-new-laravel5-project-with-homestead-quick-reference.html

Consultas complejas:
https://scotch.io/tutorials/debugging-queries-in-laravel

Migraciones con unsigned:
http://laraveldaily.com/foreign-keys-with-migrations-dont-forget-unsigned/

8.7- Mailing

http://stackoverflow.com/questions/32515245/how-to-to-send-mail-using-gmail-in-laravel-5-1
http://glob.com.au/sendmail/
https://websistent.com/using-sendmail-on-windows/
https://scotch.io/tutorials/ultimate-guide-on-sending-email-in-laravel
https://websistent.com/using-sendmail-on-windows/
https://myaccount.google.com/security

Servidores fake:
http://smtp4dev.codeplex.com
http://helibertoarias.com/es/net/smtp4dev-un-servidor-smtp-para-desarrollo/
https://prabirchoudhury.wordpress.com/2013/06/21/testing-smtp-send-email-code-on-localhost-using-smtp4dev-without-having-smtp-server-setup/

8.8- Seguridad y errores

Ausencia de salteado en tabla de usuarios:
https://mnshankar.wordpress.com/2014/03/29/laravel-hash-make-explained/

Valores para mensajes de validación:
http://stackoverflow.com/questions/17047116/laravel-validation-attributes-nice-names
https://github.com/MarcoGomesr/laravel-validation-en-espanol

Error "Invalid request (Unexpected EOF)" por consola:
http://stackoverflow.com/questions/29141240/php-local-server-invalid-request-unexpected-eof

Error "Maximum function nesting level of '100' reached, aborting!":
http://stackoverflow.com/questions/35253984/how-to-fix-laravel-5-2-this-error-maximum-function-nesting-level-of-100-reach

8.9- Análisis y requisitos

Diagramas:
http://staruml.io

Apuntes de ingeniería del software