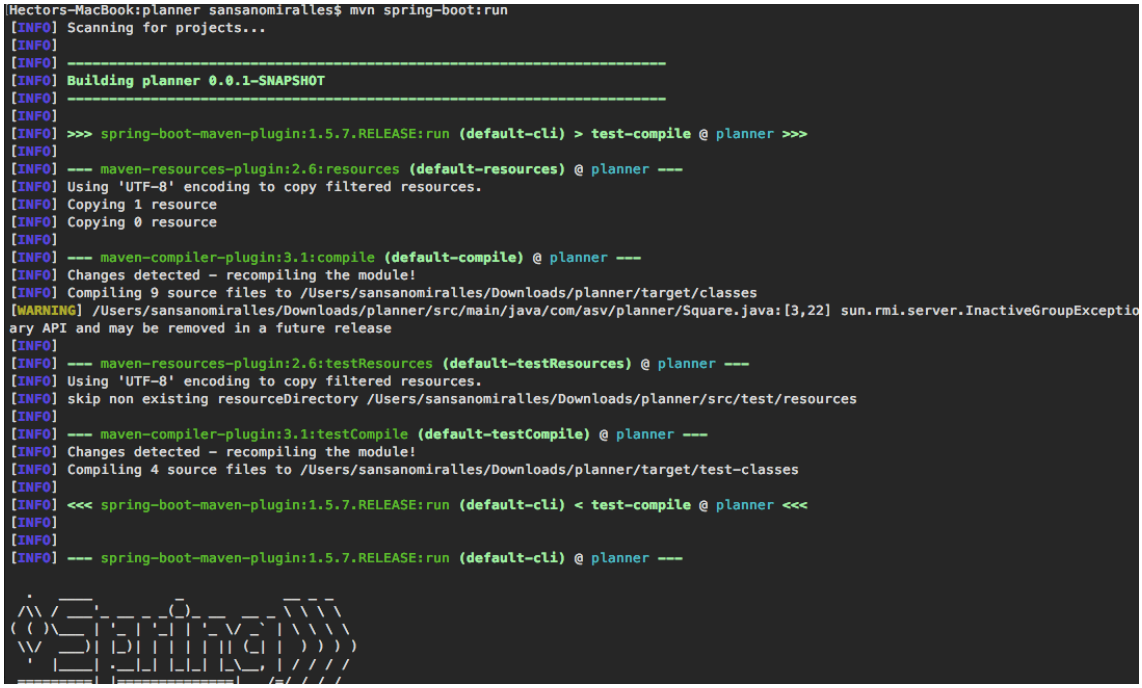
## A-Instructions to compile and make work the server and the app

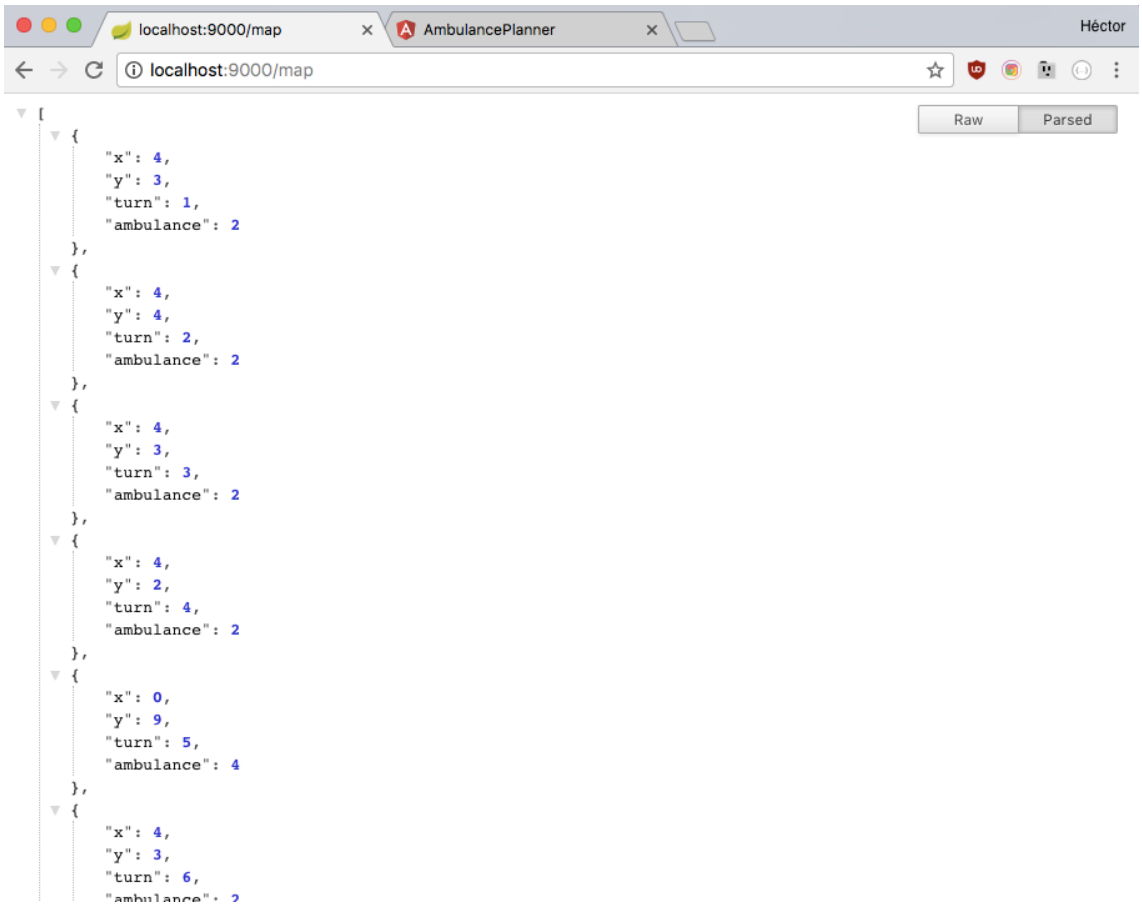
Para compilar y lanzar la aplicación de backend Springboot es necesario que JDK 7 o posterior esté instalado en la máquina. Con esta orden ya tendríamos la aplicación con su api rest funcionando.

```
mvn clean install
mvn spring-boot:run
```




Estará disponible en http://localhost:9000

Podemos lanzar una petición GET




Por el lado del cliente es necesario tener NodeJS instalado, así como el gestor de paquetes NPM.

```
npm install -g @angular/cli
```

El proyecto se ha generado con el generador de angular

```
ng init
```

Dentro de la carpeta del proyecto, instalamos las dependencias de este.

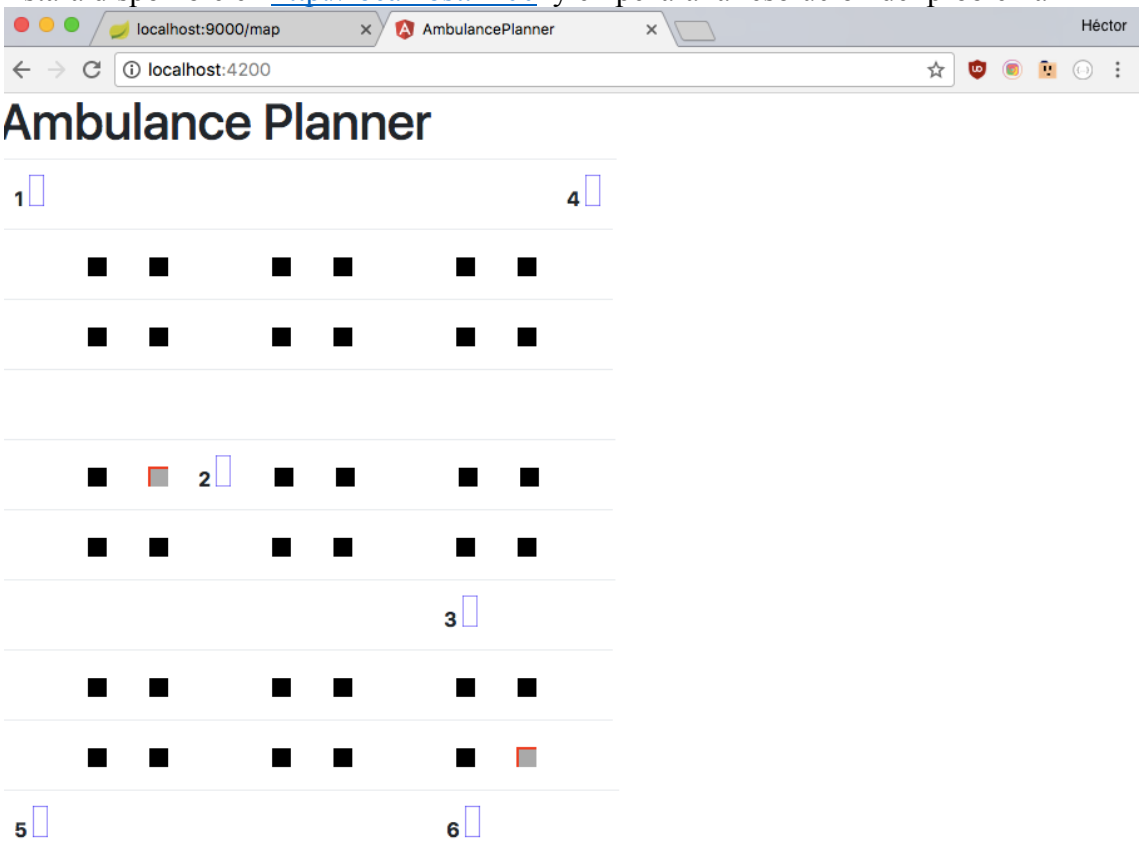
```
npm install
```

Una vez instalamos podemos lanzar la aplicación en modo desarrollo

```
ng serve
```

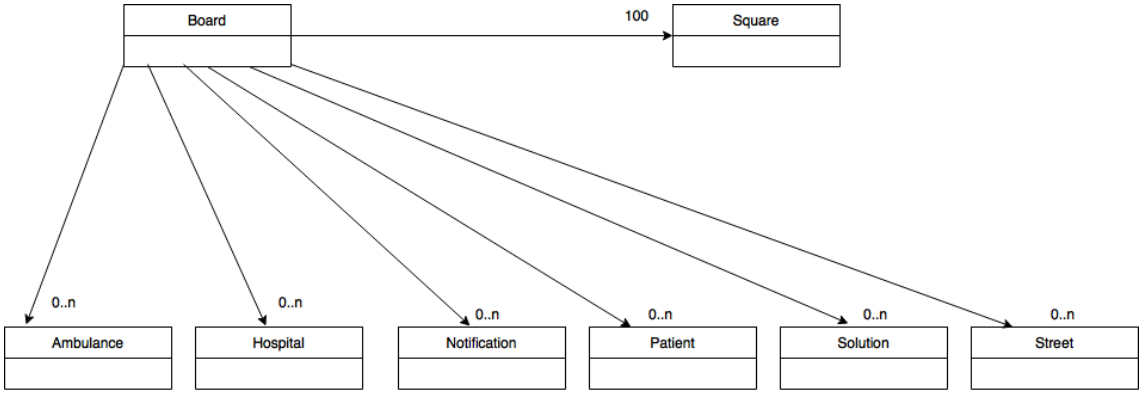
```
Hectors-MacBook:AmbulancePlanner sansanomiralles$ ng serve
As a forewarning, we are moving the CLI npm package to "@angular/cli" with the next release,
which will only support Node 6.9 and greater. This package will be officially deprecated
shortly after.

To disable this warning use "ng set --global warnings.packageDeprecation=false".

fallbackLoader option has been deprecated - replace with "fallback"
loader option has been deprecated - replace with "use"
fallbackLoader option has been deprecated - replace with "fallback"
loader option has been deprecated - replace with "use"
fallbackLoader option has been deprecated - replace with "fallback"
loader option has been deprecated - replace with "use"
fallbackLoader option has been deprecated - replace with "fallback"
loader option has been deprecated - replace with "use"
** NG Live Development Server is running on http://localhost:4200. **
Hash: bab9d77b3d919ca1beb0
Time: 25190ms
chunk    {0} polyfills.bundle.js, polyfills.bundle.map (polyfills) 234 kB {4} [initial] [rendered]
chunk    {1} main.bundle.js, main.bundle.map (main) 16.3 kB {3} [initial] [rendered]
chunk    {2} styles.bundle.js, styles.bundle.map (styles) 9.71 kB {4} [initial] [rendered]
chunk    {3} vendor.bundle.js, vendor.bundle.map (vendor) 3.29 MB [initial] [rendered]
chunk    {4} inline.bundle.js, inline.bundle.map (inline) 0 bytes [entry] [rendered]
webpack: Compiled successfully.
webpack: Compiling...
Hash: bab9d77b3d919ca1beb0
Time: 2933ms
chunk    {0} polyfills.bundle.js, polyfills.bundle.map (polyfills) 234 kB {4} [initial]
chunk    {1} main.bundle.js, main.bundle.map (main) 16.3 kB {3} [initial]
chunk    {2} styles.bundle.js, styles.bundle.map (styles) 9.71 kB {4} [initial]
chunk    {3} vendor.bundle.js, vendor.bundle.map (vendor) 3.29 MB [initial]
chunk    {4} inline.bundle.js, inline.bundle.map (inline) 0 bytes [entry]
webpack: Compiled successfully.
webpack: Compiling...
Hash: fbdf8a281f2ca02b2d87
Time: 1851ms
chunk    {0} polyfills.bundle.js, polyfills.bundle.map (polyfills) 234 kB {4} [initial]
chunk    {1} main.bundle.js, main.bundle.map (main) 16.3 kB {3} [initial] [rendered]
chunk    {2} styles.bundle.js, styles.bundle.map (styles) 9.71 kB {4} [initial]
chunk    {3} vendor.bundle.js, vendor.bundle.map (vendor) 3.29 MB [initial]
chunk    {4} inline.bundle.js, inline.bundle.map (inline) 0 bytes [entry]
```

Estará disponible en http://localhost:4200 y empezará la resolución del problema



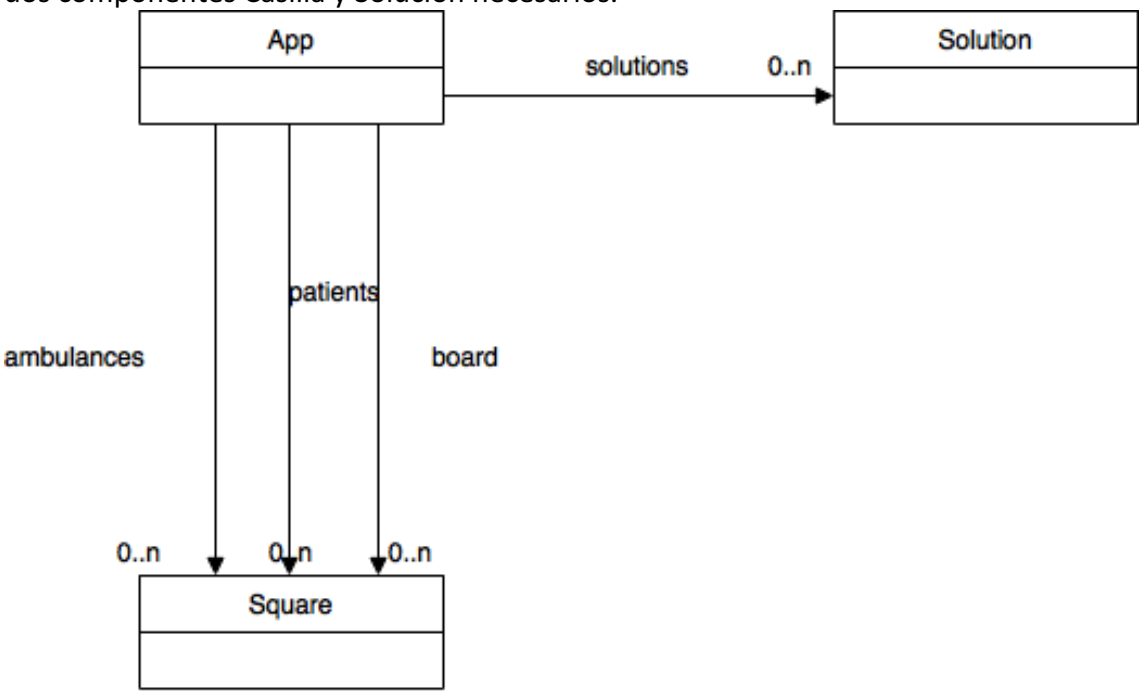
## A class diagram and a brief explanation of each one.

Para el servidor se han usado varias clases




La clase Board hace de encargada de hacer funcionar el algoritmo. Por eso está computesta de una matriz de Square que hacen de tablero. Además se tienen ArrayList de las clases Ambulance, Hospital, Notification, Patient, Solution y Street. Necesarias para resolver el problema. Estas clases podrían haber heradado de una clase Coordenada ya que todas tienen una Coordenanda X y Coordenada Y por lo menos.

En el cliente como se ha utilizado Angular2 se ha usado el componente App, y otros dos componentes Casilla y Solución necesarios.




Se aparte del tablero, se ha necesitado un Array de Ambulancias y uno de Pacientes para pintar la solución. La Solución está compuesta de la Coordenada, el instante de tiempo, y del número de ambulancia en el caso de que sea Ambulancia, si es Paciente tendrá un -1 para diferenciar

## Which design patterns you have choose and why.

Arquitectura de microservicios: Rest

Arquitectura de eventos: Angular2

GRASP

- Creador: Clase Board
- Controlador: algoritmo
- Fabricación pura: array final

GOF: Se podría haber usado el patrón Singleton en el valor del coste g(n) de la casilla.

Façade: Podríamos considerar el patrón fachada cuando hemos hecho el GET de las Soluciones y los Pacientes

## Which libraries and frameworks (including their versions) you have chosen and why

Para la parte de servidor se ha utilizado SpringBoot ya que es un framework para crear aplicaciones que usa maven. Es sencillo hacer tests unitarios y contiene el servidor Tomcat integrado, que es necesario para crear una APIRest por ejemplo. Además existen varios conectores para el acceso a datos, en esta aplicación no ha sido necesario.

Para resolver el problema se ha utilizado el Algoritmo A* principalmente para calcular el camino mínimo

Para el cliente se ha utilizado Angular2. La versión 2 ya no es un framework de Javascript, es una plataforma de desarrollo propia, además utiliza Typescript. Su orientación al modelo basado en componentes hace que el sistema esté menos acoplado y probar los componentes por separado es más sencillo desde el principio. Los componentes tienen servicios que son los que usan a su vez otros componentes.

Se ha usado un componente llamado ng2-simple-timer para hacer la función de un temporizador. Para instalarlo en nuestro proyecto con NPM

```
npm install ng2-simple-timer --save
```

También se usan los componentes Solution y Square

## Design and implementation strategy

Primero se ha comenzado por la parte del servidor, ya que iba a ser la más complicada.

- Creación de JSON del tablero y las notificaciones
- Algoritmo principal
    - Algoritmo A*
- Array de retorno

Después se ha empezado a desarrollar el cliente.

- Petición GET
- Creación del tablero
- Creación de Array de Ambulancias, Array de pacientes, Array de Resultados
- Pintar tablero en la vista
- Añadir un temporizador para representar los diferentes instantes de tiempo

## Limitations of the system

El tablero como máximo es de 10x10 casillas, aunque esto se puede cambiar.

La solución desarrollada tiene el inconveniente de no aceptar ficheros json para resolver. Se resuelve el mata que está en el código ya. Es una característica que falta.

## Potential problems found during the development

En cuanto a la elección de frameworks no resultó un problema muy grave.
Los problemas más importantes al desarrollar la aplicación ha estado en la parte de servidor. Para empezar el diseño del diagrama de clases se ha hecho un poco a ciegas. Y hasta que no se implementaba la aplicación no tenía muy claro si añadir más clases o no. El mayor problema ha sido el algoritmo que planifica las ambulancias. Se intentó hacer iterativo pero la cantidad de variables como ambulancias, pacientes, y estados de la ambulancia: ir a por el paciente, llevarlo al hospital, volver al estado inicial loc complicaron. Además solo se podía hallar un camino voraz que podría no dar la mejor solución. Se buscó un algoritmo para resolver el problema de los caminos mejor el A* y junto al método anterior se consiguió resolver el problema.
En la parte del cliente no ha habido gran dificultad.