

**Facultad de Ciencia y Tecnología**

**TRABAJO FIN DE GRADO**

Grado en Ingeniería Informática

Gestión de pistas de pádel

Alumno:

Sergio Pérez Vega

Tutores:

Beatriz Pérez Valle

**Logroño, Junio, 2017**

# Resumen

Este proyecto consiste en crear una aplicación multiplataforma para que los usuarios puedan reservar pistas de pádel de un club de pádel ficticio, en el momento que deseen y desde donde deseen de una forma fácil y sencilla.

La aplicación va a estar dividida en dos partes. Una parte dirigida al administrador de la aplicación para que pueda gestionar todos los elementos del contexto, que van a ser principalmente pistas, usuarios y reservas. La otra parte de la aplicación va dirigida a los usuarios de la propia aplicación mediante la cual podrán reservar pistas de pádel desde sus móviles o mediante un ordenador de mesa, a través de un navegador web.

En este proyecto partiremos de la premisa de que cada partido de pádel se juega con 4 jugadores, lo cual a veces no es posible. Esto hace que se reduzca el número de reservas en los clubes de pádel. Por eso, nuestra aplicación va a tener la posibilidad de reservar una pista de pádel sin contar con cuatro jugadores, para que posteriormente otros usuarios se puedan unir al partido y así sea más fácil completar un partido con cuatro jugadores de pádel.

# Abstract

For this reason, this project consists of creating a cross-platform application so that users can reserve paddle courts of a paddle club in La Rioja, whenever and wherever they want in an easy and simple way.

The application will be divided into two parts, one part of the application will be directed to the application manager so that it can manage all the elements within the application, which are going to be Courts,' Users and 'Reservations'. And the other part of the application is addressed to the users of the application itself where they can reserve paddle courts that are not reserved previously from their mobile or through the web.

The main problem that reduces the amount of paddle reserves in a paddle club is that it takes four players to play a paddle game and this is sometimes not possible. Therefore, our application will have the possibility of reserving a paddle court without having four players so that other users can join the game and thus make it easier to complete a match with four paddle players.

Contenido

[Resumen 3](#_Toc485331277)

[Abstract 3](#_Toc485331278)

[1. Introducción 6](#_Toc485331279)

[2. Motivación 6](#_Toc485331280)

[3. Planificación 6](#_Toc485331281)

[4. Tecnologías a utilizar 7](#_Toc485331282)

[5. Plan de contingencia 9](#_Toc485331283)

[6. Análisis 10](#_Toc485331284)

[6.1. Análisis de requisitos 10](#_Toc485331285)

[6.2. Diagrama de casos de uso 12](#_Toc485331286)

[7. Diseño 19](#_Toc485331287)

[7.1. Prototipos de las interfaces 20](#_Toc485331288)

[7.2. Diseño de la base de datos 25](#_Toc485331289)

[7.3. Arquitectura 27](#_Toc485331290)

[8. Implementación 27](#_Toc485331291)

[8.1. Preparación del entorno 27](#_Toc485331292)

[8.2. Preparación de la base de datos 29](#_Toc485331293)

[8.3. Principales problemas encontrados 35](#_Toc485331294)

[8.4. Exclusiones 40](#_Toc485331295)

[9. Planificación real 42](#_Toc485331296)

[10. Conclusiones 43](#_Toc485331297)

[11. Anexos 43](#_Toc485331298)

[11.1. Prototipos de las interfaces de los usuarios en la versión web 43](#_Toc485331299)

[12. Bibliografía 47](#_Toc485331300)

# Introducción

El presente TFG va a consistir en la realización de una aplicación multiplataforma que tiene como objetivo facilitar la gestión de las pistas de pádel de un club de pádel de la Rioja. En particular, se pretende que esta aplicación permita realizar la reserva de las pistas de una forma fácil e intuitiva.

Más explícitamente, los usuarios podrán reservar las pistas sin contar con cuatro personas para jugar[[1]](#footnote-1). La idea es que un usuario pueda reservar una pista de pádel en un día a una hora en concreto sin tener cuatro jugadores para jugar el partido y que la aplicación permita a los demás usuarios unirse a ese partido hasta que haya cuatro jugadores para jugarlo. Para que los jugadores disfruten más de los partidos, los jugadores estarán clasificados en la aplicación por el nivel de juego que tienen a la hora de jugar a pádel, y así, jugar contra jugadores de su mismo nivel.

El administrador podrá gestionar todos los elementos involucrados en la aplicación (reserva de pistas, usuarios, etc.). Asimismo, podrá consultar las estadísticas de uso de las diferentes pistas gestionadas por la aplicación.

El presente trabajo se ha desarrollado en la empresa Netbrain Media Solutions S.L., donde Hernán González, director de proyectos de esta empresa, actuará de cliente.

# Motivación

Este TFG nace del interés de abordar las diferentes funcionalidades citadas anteriormente a través de una aplicación que pueda ser utilizada en el futuro en algún club de pádel. En particular, para favorecer un mayor uso de las pistas de pádel, algunos clubes necesitan una aplicación móvil que permita gestionar la reserva de pistas de forma online, además de una aplicación web que la gestione.

# Planificación

A continuación se muestra la planificación que se ha realizado para lleva a cabo el presente proyecto. El total de las horas a invertir en el proyecto serán 300 horas repartidas según se indica en la siguiente tabla:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fase** | **Horas** | **Inicio** | **Fin** | **Explicación** |
| 1. **1. Planificación** | **6** | **07/02/17** | **08/02/17** | **Planificar las tareas a realizar dentro del proyecto.** |
| **2. Estudio de la tecnología** | **20** | **02/02/17** | **09/02/17** | **Estudiar y aprender a usar la tecnología a utilizar para el desarrollo del proyecto.** |
| **3. Análisis de requisitos** | **29** | **09/02/17** | **22/02/17** | **Analizar y definir los requisitos necesarios para realizar el proyecto.** |
| 3.1. Reuniones con el cliente | 4 | 09/02/17 | 09/02/17 | Acordar con el cliente los requisitos necesarios de la aplicación. |
| 3.2. Determinar los requisitos a incluir en el proyecto | 25 | 13/02/17 | 22/02/17 | Estudiar lo acordado con el cliente y decidir el alcance del proyecto. |
| **4. Diseño** | **45** | **22/02/17** | **13/03/17** | **Diseñar la funcionalidad y los prototipos de las interfaces de la aplicación.** |
| 4.1. Diseño de la base de datos | 25 | 22/02/17 | 02/03/17 | Diseño de la base de datos. |
| 4.2. Diseño de la interfaz | 20 | 06/03/17 | 13/03/17 | Diseño de los prototipos de las interfaces que va a tener la aplicación. |
| **5. Implementación** | **145** | **13/03/17** | **25/05/17** | **Implementación de la aplicación** |
| 5.1. Creación de la base de datos | 30 | 13/03/17 | 23/03/17 | Implementación de la base de datos a utilizar. |
| 5.2. Creación de las interfaces de la aplicación | 40 | 27/03/17 | 11/04/17 | Implementación de la interfaz de usuario. |
| 5.3. Implementar la funcionalidad de la aplicación | 75 | 11/04/17 | 25/05/17 | Implementar la funcionalidad necesaria de la aplicación. |
| **6. Pruebas** | **10** | **29/05/17** | **01/06/17** | **Realizar pruebas de la aplicación.** |
| **7. Redactar memoria** | **30** | **02/02/17** | **02/06/17** | **Redactar la memoria del trabajo.** |
| 7.1. Diario de lo sucedido | 9 | 02/02/17 | 02/06/17 | Usar media hora cada semana para recoger lo realizado en esa semana. |
| 7.2. Redacción de la memoria | 11 | 29/05/17 | 31/05/17 | Redactar la memoria con todo lo recogido en los anteriores pasos. |
| 7.3. Preparación de la presentación del proyecto | 10 | 31/05/17 | 02/06/17 | Realización y estudio de la presentación para la defensa del proyecto. |
| **8. Reuniones con la tutora de la Universidad** | **15** | **02/02/17** | **02/06/17** | **Realizar reuniones periódicas con la tutora de la Universidad.** |

# Tecnologías a utilizar

Las tecnologías usadas para la realización de este proyecto son las siguientes:

* *Angular 2* (también llamado *AngularJS)*: es un framework de *Javascript* de código abierto, mantenido por Google. Su objetivo es favorecer el uso del patrón *MVC* (Modelo Vista Controlador) para aplicaciones web, que sirve para hacer que el desarrollo sea más fácil y reducir la manipulación del *DOM* (Document Object Model: conjunto estándar de objetos para representar documentos *HTML*). Este framework amplia el *HTML* para añadir contenido dinámico que permite la sincronización bidireccional automática entre modelos y vistas.

Hemos elegido esta tecnología porque facilita la bidireccionalidad entre los modelos y las vistas y porque es un framework que está siendo muy utilizado para la realización de aplicaciones web. Ha sido usado principalmente para el paso de información de la capa de presentación con la capa de lógica de negocio.[3,4]

* *Ionic 2*: es un *framework* para aplicaciones móviles. Es un *SDK* completo de código abierto para aplicaciones móviles híbridas. Construido en *AngularJS* y *Apache Cordova* proporciona herramientas y servicios para aplicaciones móviles híbridas usando tecnologías web como *HTML5* y *CSS*. Es el framework utilizado para el desarrollo y la compilación del proyecto en las diferentes plataformas a utilizar.

Hemos elegido esta tecnología porque nos permite escribir el código de la aplicación una sola vez y compilar ese mismo código en diferentes plataformas. Además, siempre te permite añadir nuevas plataformas en caso de que el proyecto siga aumentando y sea necesario compilar el código para ser usado en otra plataforma. También la hemos elegido porque utiliza *AngularJS*, que coincidimos en que era una tecnología muy utilizada actualmente y que puede ser de gran ayuda en un futuro. Aunque también tiene la pega, de que, al no ser una tecnología nativa, en un principio es una plataforma menos eficiente en cuanto a la velocidad de las aplicaciones, pero decidimos darle menos importancia a este aspecto debido a que nuestra aplicación no va a tener grandes gráficos en los que sí se notase más esta pega.[1,2]

* *Node JS*: es un entorno en tiempo de ejecución multiplataforma, de código abierto basado en el motor V8 de Google. Usa un modelo con un único hilo de ejecución con operaciones E/S sin bloqueo y orientado a eventos. Está diseñado para construir aplicaciones en red escalables. Usado para la instalación de *Ionic 2*.[7]
* *Typescript*: es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de *JavaScript*, que añade tipado estático y objetos basados en clases. A través de un compilador de *Typescript* se traduce el código *Typescript* en código *Javascript*. Este compilador está escrito en el propio *Typescript*. Está pensado para grandes proyectos. Elegimos esta tecnología porque es la principal tecnología utilizada en proyectos con *Ionic2*. Usado para la implementación de la capa de lógica de negocio.[5,6]
* *Firebase*: es una plataforma de desarrollo para aplicaciones móviles y web que utiliza la infraestructura de *Google*. Se compone de características que los desarrolladores pueden mezclar y combinar para ajustarlo a sus necesidades. El producto inicial de *Firebase* era una base de datos en tiempo real que permitía a los desarrolladores almacenar y sincronizar los datos a través de múltiples clientes. Con el tiempo se ha ampliado el conjunto de productos que ofrece para convertirse en un producto completo para desarrolladores. De los diferentes productos que ofrece este, en el proyecto se van a usar:
  + *Realtime Database*: proporciona una base de datos en tiempo real que almacena los datos en una base de datos *NoSQL*. Los datos son almacenados con formato *JSON*. También proporciona herramientas para el acceso y la modificación de los datos de la base de datos. Usada para almacenar los datos necesarios para el proyecto.
  + *Firebase Auth*: proporciona diferentes herramientas para controlar la autenticación de los usuarios dentro de la aplicación. Para ello nos proporciona diferentes posibilidades de autenticación. En particular, en este proyecto se utilizará la autenticación mediante email y contraseña.

Elegimos esta tecnología, principalmente porque es un servicio de *Google*, lo que da bastantes garantías y es un servicio ya probado y en nuestra opinión más fiable que uno realizado por nosotros mismos. También nos ahorraba el tener que crear nosotros nuestro propio servicio. Es altamente accesible y nos permite diferentes servicios para facilitar la implementación del código, como por ejemplo, el servicio de autenticación. Por último, es fácilmente escalable en caso de que nuestra aplicación siga creciendo, porque de esta parte se encarga *Google*.[16,17]

* *Visual Studio Code*: es un entorno de desarrollo integrado para sistemas operativos Windows. Este entorno soporta múltiples lenguajes de programación. En particular, es este proyecto se ha utilizado para desarrollar con *Ionic 2, AngularJS, Typescript, HTML5* y *CSS*.[11]
* *Android Studio*: entorno de desarrollo integrado oficial para el desarrollo de aplicaciones para Android basado en *IntelliJ IDEA*. Tiene un sistema de compilación basado en *Gradle* y permite emular las aplicaciones desarrolladas en este entorno desde dispositivos móviles conectados al ordenador por *USB* sin necesidad de compilar un nuevo *APK* (es el archivo ejecutable de las aplicaciones en sistemas *Android*). Usado para una vez terminado el proyecto en *Ionic 2* y compilado para la plataforma *Android*, lo utilizaremos para generar el archivo *APK* ejecutable en un dispositivo android.[8]
* *Github*: es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones *Git*. Permite almacenar el código de forma pública aunque también se puede hacer de forma privada creando una cuenta de pago. En nuestro caso se ha utilizado para alojar el proyecto y para controlar las versiones del mismo. En particular, se ha usado Github para alojar el desarrollo de este proyecto (ver enlace en [13]).
* *Balsamiq mockups*: es una aplicación que permite diseñar prototipos de interfaces. Usada para el diseño de los prototipos de las interfaces del proyecto creados en la fase de diseño. [14]
* *ArgoUML*: es una aplicación que permite diseñar diagramas *UML*, en particular los diagramas de Casos de Uso y Diagramas de clases creados en la fase de análisis. [15]
* Además de todas las tecnologías anteriores también se ha utilizado *CSS* y *HTML5*. [9,10]

# Plan de contingencia

Los principales riesgos que podrían tener lugar durante el desarrollo del presente proyecto, su nivel de impacto y la forma de intentar corregirlos son los siguientes:

*Riesgo*: Cambio de requisitos.

* *Impacto*: Dependerá de la fase en la que se encuentre el proyecto. Cuanto más avanzado esté el proyecto mayor impacto negativo tendrá.
* *Nivel de probabilidad de que ocurra*: Medio.
* *Fases en las que se puede producir*: Todas las fases.
* *Plan de contingencia*: Dejar bien claro en el análisis de requisitos el alcance del proyecto. En el caso de que el cliente sugiera alguna modificación, estudiar la posibilidad de llevarla a cabo o no.

*Riesgo*: Cambios de la tecnología o problemas con la misma.

* *Impacto*: Muy alto.
* *Nivel de probabilidad de que ocurra*: Alto.
* *Fases en las que se puede producir*: Fase de implementación.
* *Plan de contingencia*: Establecer antes de empezar la implementación cuál es la mejor tecnología a utilizar, una vez elegida, estudiarla. En el caso de descubrir que no se consigue avanzar con la tecnología escogida, dependiendo de la fase en la que nos encontremos. Si estamos en las primeras fases, elegir otra tecnología parecida a la elegida y continuar con ella. Pero si estamos en las últimas fases, llegar a un consenso con el cliente para intentar suplir los problemas.

*Riesgo*: Problemas derivados de una mala planificación.

* *Impacto*: Alto.
* *Nivel de probabilidad de que ocurra*: Medio.
* *Fases en las que se puede producir*: Todas las fases.
* *Plan de contingencia*: Crear una planificación lo más realista posible. En caso de no poder cumplir la planificación, procurar reajustar la planificación en el momento en el que nos demos cuenta.

*Riesgo*: Problemas personales que impiden realizar el trabajo en las horas asignadas para realizarlo.

* *Impacto*: Alto.
* *Nivel de probabilidad de que ocurra*: Medio.
* *Fases en las que se puede producir:* Todas las fases.
* *Plan de contingencia*: Intentar que no ocurra, y en el caso de que dichos problemas tengan lugar, conseguir tiempo no incluido en la planificación (como por ejemplo algún fin de semana) con objeto de recuperar dichas horas.

*Riesgo*: Pérdida de la documentación.

* *Impacto*: Muy alto.
* *Nivel de probabilidad de que ocurra*: Bajo.
* *Fases en las que se puede producir*: Todas las fases.
* *Plan de contingencia*: Creación de copias de seguridad de toda la información correspondiente al proyecto en diferentes dispositivos. En nuestro caso se utilizará un disco duro externo, un *USB* y *Google Drive*. Además, como se ha comentado anteriormente, se almacenará la información en *GitHub*.

# Análisis

Se describe en este capítulo el análisis de requisitos, el diagrama de casos de uso y la especificación de los casos de uso.

### Análisis de requisitos

A continuación describiremos el análisis de requisitos de la aplicación, los cuales han sido establecidos por el cliente. Se distingue entre requisitos funcionales y requisitos no funcionales.

* + 1. Requisitos funcionales

Dividiremos estos requisitos en función de si refieren a la parte gestionada por el administrador o a la parte correspondiente a los usuarios (tanto visitantes como registrados). Ambas partes dispondrán de una versión móvil y web.

En la aplicación para el administrador en ambas versiones (versión móvil y versión web):

* El administrador podrá autenticarse para acceder a la aplicación.
* El administrador podrá ver la disponibilidad de las pistas de pádel sin reserva. Desde aquí podrá seleccionar una pista a una hora y un día en concreto para reservarla en nombre de un usuario.
* El administrador podrá gestionar los usuarios creando nuevos y viendo la lista de los ya creados. Desde la lista de usuarios podrá modificar los datos personales del usuario que desee y también podrá eliminarlo.
* El administrador podrá gestionar las pistas del club de pádel creando pistas nuevas y viendo la lista de las ya creadas. Desde la lista de las pistas podrá eliminar la pista que desee.
* El administrador podrá gestionar las reservas de las pistas por los usuarios dentro de la aplicación viendo la lista de las reservas ya realizadas. Desde esta lista, podrá eliminar la reserva que desee.
* El administrador podrá ver estadísticas sobre el uso de las pistas y sobre las reservas.

En la aplicación para los usuarios visitantes en ambas versiones:

* Todo usuario visitante, si así lo desea, se podrá registrar en la aplicación para poder usarla proporcionando como datos el nombre, el DNI, el teléfono, el nivel de juego, un nombre de usuario, la contraseña, si es abonado del club de pádel o no y el email.

Respecto a los usuarios registrados, distinguiremos entre la versión web y la versión móvil:

* En la versión web:
* El usuario podrá autenticarse para acceder a la aplicación en el momento que desee.
* El usuario podrá modificar sus datos personales en cualquier momento.
* El usuario podrá ver en todo momento la disponibilidad de las pistas sin reservar. Desde aquí, el usuario podrá reservar una pista en el momento que desee siempre que la pista no esté ya reservada.
* El usuario podrá ver el historial de reservas que ha realizado.
* El usuario podrá elegir si quiere recibir notificaciones o no.
* En la versión móvil, los usuarios registrados tendrán los mismos requisitos que en la versión web, añadiendo a estos requisitos:
* Cuando un usuario registrado reserve una pista con un número de jugadores inferior a 4, el sistema enviará una notificación a todos los usuarios del mismo nivel que el usuario que ha reservado la pista y que tengan activas las notificaciones, de forma que estos puedan unirse al partido.

Nota: La idea inicial era realizar el envío de notificaciones en la versión móvil de los usuarios registrados y por lo tanto los siguientes pasos del análisis incluyen este requisito. Pero como ya explicaré más adelante este requisito no se ha podido implementar debido a dificultades con la tecnología.

* + 1. Requisitos no funcionales

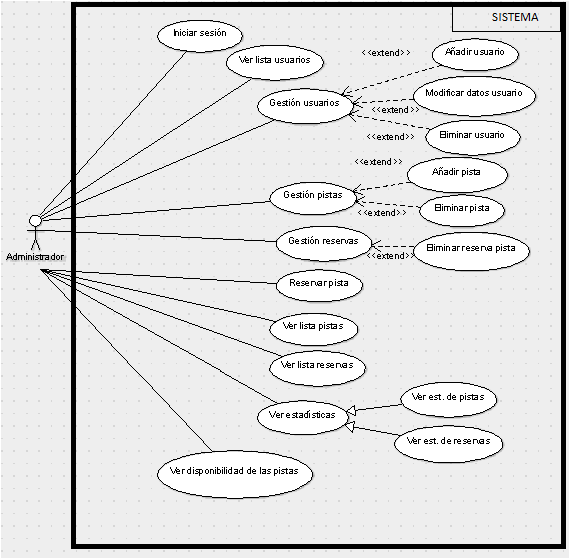
En este apartado se presentan los requisitos no funcionales:

* La interfaz tanto web como móvil será fácil e intuitiva con objeto de facilitar el uso a los usuarios y al administrador.
* La aplicación móvil deberá funcionar en Android versión 6.0.1 y la aplicación web funcionará en los navegadores *Google Chrome* versión 58.0.3y *Mozilla Firefox* versión 53.0.3.
* La aplicación en todas sus versiones podrá usar algunas funcionalidades en modo offline (sin acceso a internet)

### Diagrama de casos de uso

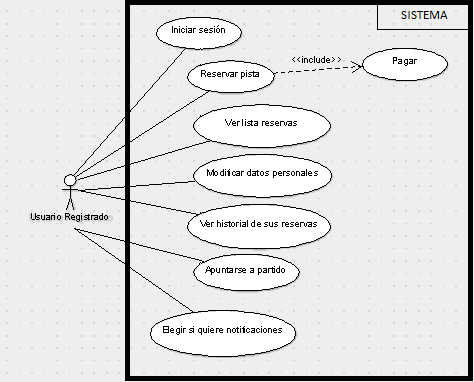
En base a los anteriores requisitos, se han identificado tres actores que interactúan con la aplicación que van a ser *Administrador*, *Usuario visitante* y *Usuario registrado*. A continuación se presentan los Diagramas de casos de uso.

El diagrama de casos de uso de la aplicación tanto web como móvil para el administrado es:



El diagrama de casos de uso del actor *Usuario visitante* solamente contiene un Caso de Uso que es el de Registrarse.

Y el diagrama de casos de uso del actor *Usuario registrado*:



* + 1. Especificación de los casos de uso

A continuación, se explica cada uno de los casos de uso presentados en el apartado anterior:

|  |
| --- |
| **Caso de Uso:** Iniciar sesión. |
| **Objetivo:** Iniciar sesión en la aplicación con su nombre de usuario y contraseña. |
| **Actor(es):** Administradory Usuario registrado. |
| **Precondición:** El administrador o el Usuario registrado deben estar dados de alta en la aplicación. |
| **Pasos:**   1. El administrador o usuario registrado accede a la aplicación. 2. Introduce el nombre de usuario y contraseña. 3. Accede a la aplicación. |
| **Extensiones:**   * 1. Introduce un nombre de usuario que no exista ya en la aplicación.      1. El sistema le avisa de que el usuario introducido no existe.   2. Introduce una contraseña no válida.      1. El sistema le avisa de que está introduciendo una contraseña que no es válida para ese nombre de usuario. |

|  |
| --- |
| **Caso de Uso:** Añadir usuario. |
| **Objetivo:** Añadir un usuario nuevo a la aplicación para que pueda utilizarla con su nombre de usuario y contraseña. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección de gestión de usuarios. 2. Pulsa en el botón de añadir un usuario. 3. Añade los datos del nuevo usuario. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Eliminar usuario. |
| **Objetivo:** Se eliminará un usuario de la aplicación. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección de gestión de usuarios. 2. Elige usuario deseado. 3. Elimina el usuario deseado. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Añadir pista. |
| **Objetivo:** Añadir una pista nueva a la aplicación para que pueda ser reservada y utilizada. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección de gestión de pistas. 2. Pulsa en el botón de añadir una pista. 3. Crea una pista nueva. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Eliminar pista. |
| **Objetivo:** Eliminar de la aplicación una pista ya existente. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado y que la pista esté dada de alta en la aplicación |
| **Pasos:**   1. El administrador accede a la sección de gestión de pistas. 2. Elige la pista deseada. 3. Elimina la pista deseada. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Reservar pista. |
| **Objetivo:** Reservar una pista para un usuario. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado y que la pista no esté reservada en el día y la hora deseada. |
| **Pasos:**   1. El administrador accede a la sección de ver la disponibilidad de las pistas. 2. Elige la hora, el día y la pista a reservar. 3. Introduce el nombre del usuario que la quiere reservar. 4. Reserva la pista. |
| **Extensiones:**   * 1. Introduce un nombre de usuario que no exista en la aplicación.      1. El sistema le avisa de que el nombre de usuario introducido no existe. |

|  |
| --- |
| **Caso de Uso:** Eliminar reserva pista. |
| **Objetivo:** Eliminar una reserva ya realizada. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado y que la pista esté reservada a la fecha y hora indicada. |
| **Pasos:**   1. El administrador accede a la sección de gestión de reservas. 2. Elige la reserva deseada. 3. Elimina la reserva de la pista. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Ver disponibilidad de las pistas. |
| **Objetivo:** Ver la disponibilidad de las pistas sin reservar en los días deseados. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección para ver la disponibilidad de las pistas. 2. Ve la disponibilidad de las pistas en un día en concreto. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Ver lista reservas. |
| **Objetivo:** Ver todas las reservas que hay realizadas de las pistas. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección de gestión de reservas. 2. Ve la lista de reservas realizadas por los usuarios en un día concreto. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Ver lista usuarios. |
| **Objetivo:** Ver todos los usuarios que hay dados de alta en la aplicación. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección de gestión de usuarios. 2. Ve la lista de usuarios dados de alta. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Modificar datos de usuario. |
| **Objetivo:** Modificar los datos de un usuario ya dado de alta en la aplicación. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado y que el usuario que se va a modificar ya exista. |
| **Pasos:**   1. El administrador accede a la sección de gestión de usuarios. 2. Ve la lista usuarios. 3. Elige el usuario deseado. 4. Modifica los datos del usuario elegido. 5. Acepta cambios. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Ver lista pistas. |
| **Objetivo:** Ver el listado de las pistas de pádel. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a la sección de gestión de pistas. 2. Ve la lista de pistas de pádel. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Ver estadísticas de pistas. |
| **Objetivo:** Ver estadísticas sobre el uso de las pistas de pádel. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede a las estadísticas. 2. Ve las estadísticas de las pistas. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Ver estadísticas de reservas. |
| **Objetivo:** Ver estadísticas sobre el uso de las pistas de pádel. |
| **Actor(es):** Administrador. |
| **Precondición:** El administrador debe haberse autenticado. |
| **Pasos:**   1. El administrador accede las estadísticas. 2. Ve las estadísticas de las reservas. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Registrarse. |
| **Objetivo:** Usuario sin registrar se registre en la aplicación para poder utilizarla. |
| **Actor(es):** Usuario visitante. |
| **Precondición:** |
| **Pasos:**   1. Visita la aplicación sin ser usuario registrado. 2. Accede a la interfaz de registro. 3. Introduce los datos personales del usuario nuevo. 4. Se registra. |
| **Extensiones:**   * 1. Introduce un usuario que no exista ya en la aplicación.      1. Si introduce un usuario ya existente el sistema le avisa de que no puede introducir ese nombre de usuario.   2. Introducir una contraseña válida.      1. Si introduce una contraseña inválida el sistema le avisa de que no puede introducir esa contraseña.   3. Todos los demás campos son obligatorios.      1. Si no introduce valores en los campos requeridos o introduce valores no válidos se le indicará que no puede introducir esos valores. |

|  |
| --- |
| **Caso de Uso:** Reservar pista. |
| **Objetivo:** Reservar una pista en una fecha y hora deseadas. |
| **Actor(es):** Usuario registrado. |
| **Precondición:** El usuario registrado debe estar autenticado. |
| **Pasos:**   1. El usuario registrado accede a la sección de reservas. 2. Elige la pista deseada. 3. Elige la fecha y hora deseada. 4. Reserva la pista. |
| **Extensiones:**   * 1. La pista ya está reservada ese día a esa hora.      1. El sistema le avisa que esa pista en ese momento ya está reservada y que no puede reservarla.   2. Si el usuario introduce un número de jugadores inferior a 4 el sistema enviará una notificación a aquellos usuarios del mismo nivel que el usuario registrado y que tengan activadas las notificaciones.      1. Los usuarios que reciben la notificación pueden unirse al partido o rechazarlo. |

|  |
| --- |
| **Caso de Uso:** Ver disponibilidad de las pistas. |
| **Objetivo:** Ver la disponibilidad de las pistas en una fecha y hora deseada. |
| **Actor(es):** Usuario registrado. |
| **Precondición:** El usuario registrado debe estar autenticado. |
| **Pasos:**   1. El usuario registrado accede a ver la disponibilidad de las pistas. 2. Elige la fecha deseada. 3. Ve la disponibilidad de las pistas en la fecha deseada. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Modificar datos personales. |
| **Objetivo:** Modificar los datos personales del usuario. |
| **Actor(es):** Usuario registrado. |
| **Precondición:** El usuario registrado debe estar autenticado. |
| **Pasos:**   1. El usuario registrado accede a ver sus datos personales. 2. Modifica los datos. 3. Guarda los datos modificados. |
| **Extensiones:**   * 1. Introduce valores inválidos.      1. El sistema le avisa de que está introduciendo datos inválidos. |

|  |
| --- |
| **Caso de Uso:** Ver historial de sus reservas. |
| **Objetivo:** Ver el historial de todas las reservas realizadas por el usuario registrado. |
| **Actor(es):** Usuario registrado. |
| **Precondición:** El usuario registrado debe estar autenticado. |
| **Pasos:**   1. El usuario registrado accede a la sección del historial de sus reservas realizadas. 2. Ve la lista de reservas realizadas por él. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Elegir si quiere notificaciones. |
| **Objetivo:** Elegir si quiere o no recibir notificaciones de la aplicación. |
| **Actor(es):** Usuario registrado. |
| **Precondición:** El usuario registrado debe estar autenticado. |
| **Pasos:**   1. El usuario registrado accede a la sección de modificación de sus datos personales. 2. Selecciona si quiere notificaciones o no. 3. Guarda los cambios realizados. |
| **Extensiones:** |

|  |
| --- |
| **Caso de Uso:** Apuntarse a un partido. |
| **Objetivo:** Apuntarse a un partido que ya haya sido creado por otro usuario de su mismo nivel. |
| **Actor(es):** Usuario registrado. |
| **Precondición:** El usuario registrado debe estar autenticado y el partido ha de estar creado por otro usuario de su nivel. |
| **Pasos:**   1. Cuando el usuario registrado recibe la notificación pincha en ella para acceder a la información del partido. 2. Acepta la invitación al partido. |
| **Extensiones:** |

# Diseño

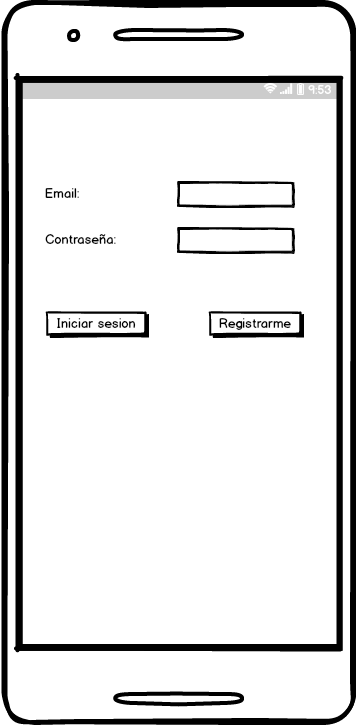
En esta sección describiremos los aspectos relacionados con la fase de diseño de nuestra aplicación. En particular, se mostrará el diseño de las interfaces de usuario y el diseño de la base de datos.

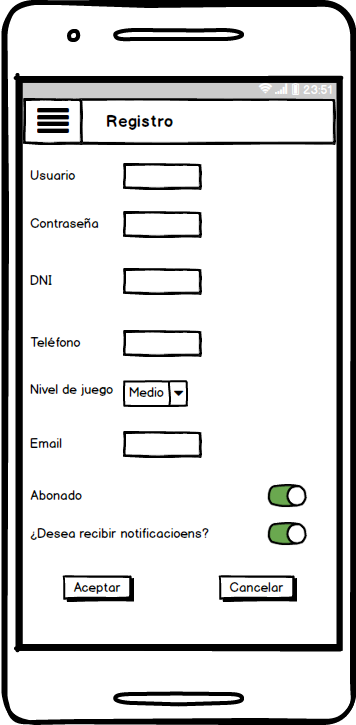
### Prototipos de las interfaces

En este apartado se muestran los prototipos de las interfaces de usuario que va a tener la aplicación (tanto web como móvil). Se han dividido en dos partes, que a su vez están divididas en otras dos partes. Primero se diferencia entre interfaces para usuarios y para el administrador, y cada uno de los mismos tendrá la versión móvil y la versión web.

* + 1. Interfaces de la versión móvil para usuarios registrados

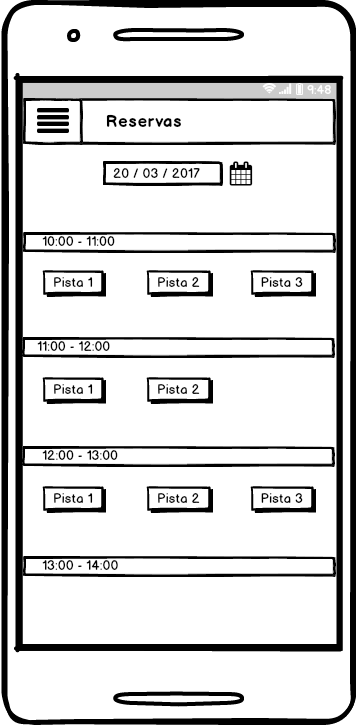
La primera pantalla que aparecerá al acceder a la aplicación, en el caso de que el usuario no haya iniciado sesión, será la que se muestra en la imagen de la izquierda. En caso de que el usuario no tenga una cuenta creada, podrá seleccionar el botón “Registrarme” desde el que accederá a la pantalla de la derecha, que es la pantalla de registro donde podrá introducir sus datos y así crear un nuevo usuario.





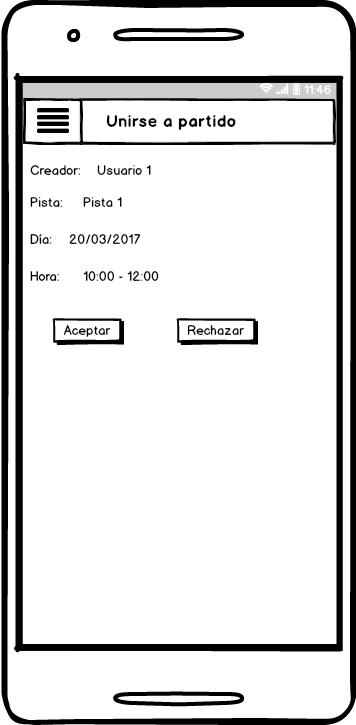
Cuando el *usuario registrado* se haya autenticado, verá directamente la interfaz que implementa la funcionalidad principal de la aplicación, es decir, la de realización de reservas, donde podrá ver la lista de las pistas que no estén reservadas considerando la fecha actual por defecto (ver la imagen situada a la izquierda en la siguiente pantalla).

Además, la aplicación dispondrá de un menú desplegable en la parte izquierda superior de la pantalla con el que podremos acceder a todas las demás interfaces (ver imagen de la derecha).

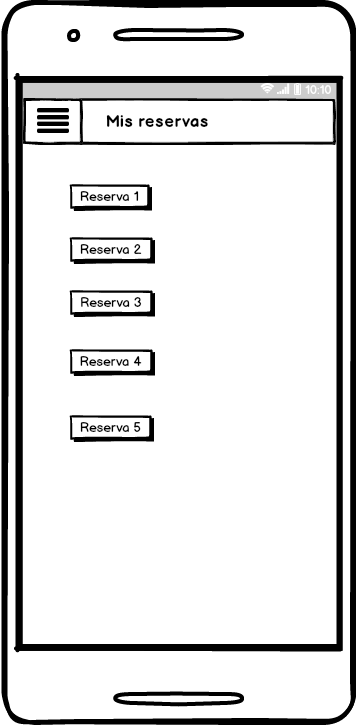
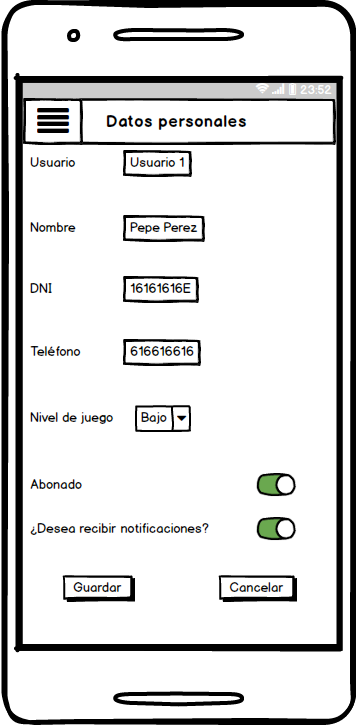
Cuando el usuario quiera reservar una pista a una hora y un día, elegirá en la pantalla principal de la aplicación el día que desee e indicará la hora deseada. Posteriormente, pinchará sobre la pista que quiere reservar siempre y cuando esta pista aparezca en esa hora y día. Si no aparece, significará que ya está reservada. En ese momento, la aplicación llevará al usuario a la pantalla en la que se reserva la pista (ver la imagen de la izquierda de las siguientes imágenes). En esta pantalla aparecerán ya rellenos los campos con el nombre de usuario, el nombre de la pista, el día y la hora de inicio de la reserva; estos datos, el usuario, no los podrá modificar. El usuario sólo podrá modificar el número de jugadores abonados y no abonados y si finalmente desea reservar la pista le dará al botón de “Reservar”.

Si el usuario ha introducido un número de jugadores inferior a 4, el sistema enviará a todos los usuarios que tengan activadas las notificaciones y el mismo nivel de juego que el usuario que está reservando una notificación para que estos usuarios puedan unirse al partido hasta que el mismo esté completo. Cuando el usuario acceda a esa notificación accederá a la interfaz para unirse al partido (ver la imagen de la derecha).



Los usuarios de la aplicación podrán modificar sus datos personales accediendo a la interfaz de datos personales desde el menú desplegable de arriba a la izquierda (ver imagen de la izquierda de las imágenes siguientes).

Los usuarios también podrán ver un historial con todas las reservas que han realizado en la aplicación (ver imagen de la derecha de las siguientes imágenes).



* + 1. Prototipos de las interfaces de la versión web para usuarios registrados

Esta versión tiene las mismas pantallas que en la versión móvil, con excepción de la pantalla para unirse a los partidos, que como se ha comentado anteriormente sólo va a aparecer en versiones móvil.

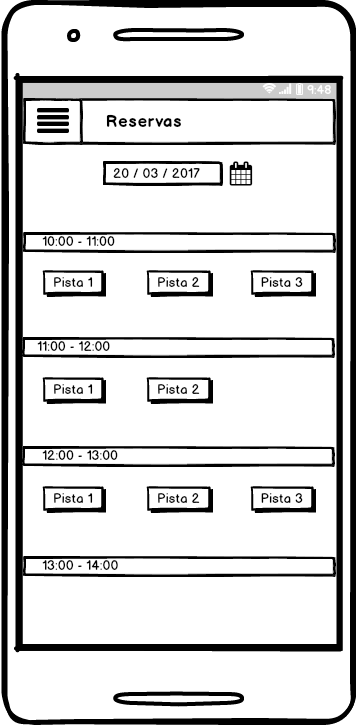
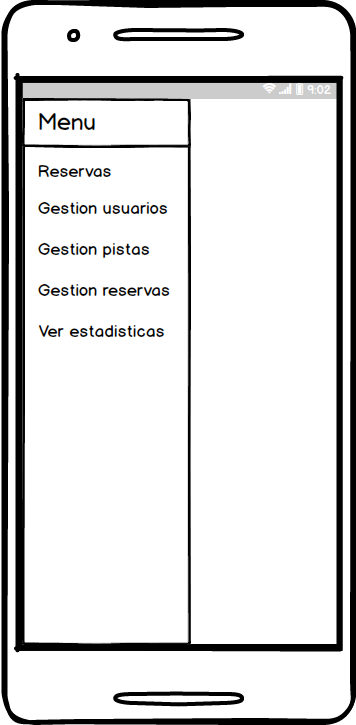
Estas interfaces se pueden ver en el anexo que hay al final del documento.

* + 1. Prototipos de interfaces de la versión móvil y web para el administrador

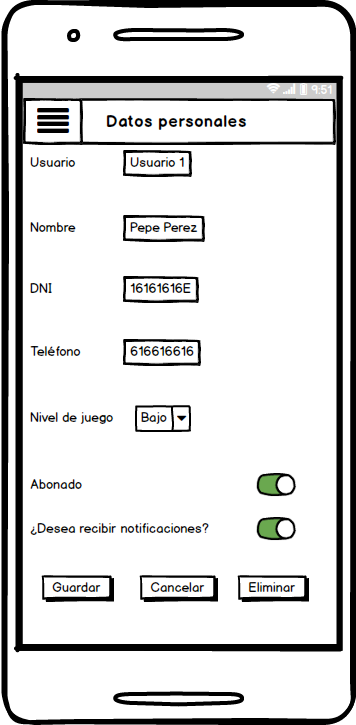
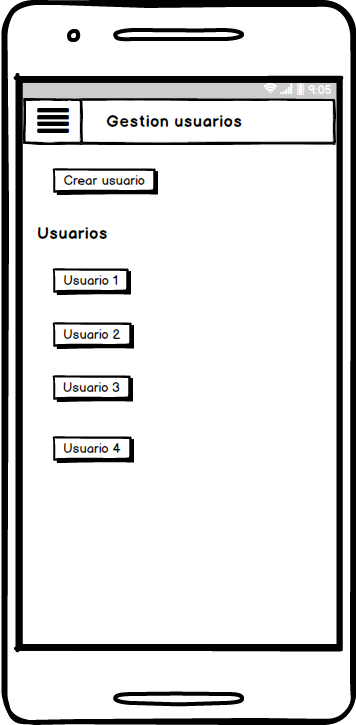
Ambas versiones del administrador van a tener las mismas interfaces con la única diferencia de que unas van a ser en versión móvil y otras en versión web, pero ambas van a tener la misma estructura y los mismos elementos. El administrador va a tener una cuenta ya creada donde sólo se guardará el email y la contraseña con la que va a acceder a la aplicación, por lo tanto, no habrá ninguna interfaz de registro para que el administrador se pueda registrar. Debido a esto, el administrador accederá a la aplicación, y en caso de no estar ya autenticado, le aparecerá la misma interfaz que la que le aparece al usuario para iniciar sesión.

Una vez que el administrador haya iniciado sesión, los prototipos de las interfaces que le aparecerán serán las siguientes:

Esta aplicación tendrá un menú desplegable al igual que el de la aplicación para usuarios con diferentes enlaces a sus correspondientes funcionalidades (ver imagen de la izquierda). Además de esto, la interfaz principal de la aplicación será la misma que la que aparecerá en la aplicación para usuarios (ver imagen de la derecha). Esta interfaz es la misma que la interfaz de la versión de usuarios con la diferencia de que cuando pulse en una pista para reservarla, el administrador tendrá que introducir el nombre de usuario que tenga el usuario que desea reservar la pista. Por este motivo, el nombre de usuario de cada usuario registrado tiene que ser único.



El administrador puede gestionar los usuarios, las reservas y las pistas accediendo desde el menú desplegable mostrado anteriormente. Cuando el administrador acceda a la “Gestión de usuarios” verá la lista de usuarios registrados en la aplicación (ver imagen de la izquierda), El administrador podrá pulsar sobre un usuario para ver sus datos personales, modificarlos o incluso eliminar el usuario (ver imagen de la derecha).

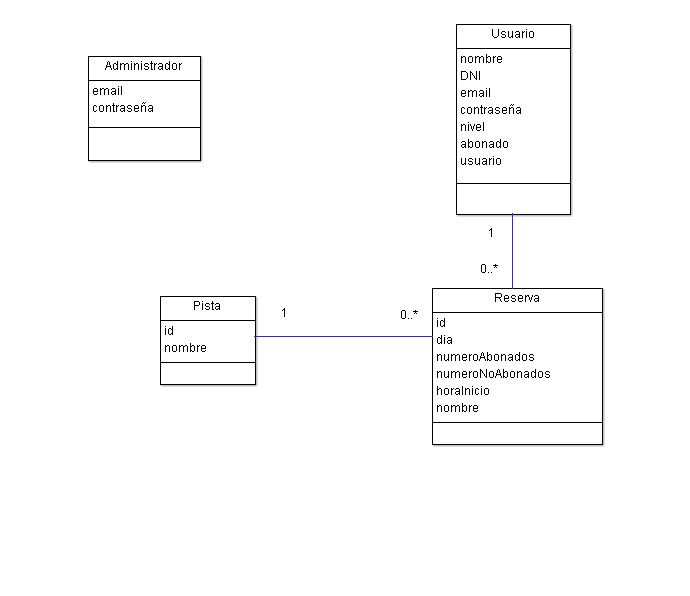


Las interfaces de gestión de pistas y de reservas son similares a la de gestión de usuarios con la diferencia de que en la de gestión de reservas la lista será una lista con las reservas que hay realizadas y en la gestión de pistas habrá una lista con las pistas creadas en la aplicación. Además, al pulsar una pista o reservas, el administrador también podrá eliminar esa pista o reserva.

Por último, en la interfaz para ver las estadísticas de la aplicación, el administrador verá cuales son las pistas más reservadas y cuáles son las horas con más reservas.

### Diseño de la base de datos

Se ha decidido diseñar un diagrama de clases que simule el esquema de lo que sería la BD relacional que almacenaría los datos de la aplicación, con objeto de ayudarnos a definir el esquema del JSON, que como hemos dicho anteriormente, es la forma en la que se almacena la información en *Firebase*.



En la clase “Administrador” sólo se va a almacenar un email y una contraseña que son el email y la contraseña con la que el administrador se va a autenticar en la aplicación. Está clase no está unida con ninguna otra puesto que lo único que va ha hacer el administrador es gestionar todas las demás.

En la clase “Usuario” se va a almacenar también un email y contraseña, que son los valores necesarios para que los usuarios se autentiquen en la aplicación. También se almacena un nombre de usuario que es el campo llamado “usuario”, que va a ser único y diferente para cada usuario. Y luego se va a almacenar información personal del usuario para que el administrador pueda tener información sobre el usuario.

La clase “Reserva”, esta relacionada con la clase “Usuario” de forma que un usuario puede tener ninguna o muchas reservas y una reserva solo puede estar realizada por un usuario únicamente. Por esto, además de los atributos de la propia clase “Reserva” también va a tener otro atributo derivado de esa relación, que va a ser el nombre de usuario del usuario que realice la reserva para poder identificar a que usuario corresponde cada reserva. Esta clase también está relacionada con la clase “Pista” de forma que una pista puede tener muchas reservas y una reserva solo puede ser de una pista, por lo tanto, la clase “Reserva” tiene otro atributo derivado de esta relación que es el “id” de la clase “Pista” para identificar la pista que se ha reservado en cada reserva.

En la clase “Pista” solo se almacena el identificador único para cada pista y su nombre.

### Arquitectura

La arquitectura usada en este proyecto es la arquitectura recomendada por la tecnología a usar, que es el patrón MVC (modelo, vista, controlador). Por lo tanto, es un modelo que separa el código en tres capas, la capa de datos, la capa de lógica de negocio y la capa de presentación. En este caso, la capa de datos corresponde con el modelo, la lógica de negocio con el controlador y la capa de presentación con la vista.

La capa de datos es la que se encarga de recoger y enviar los datos necesarios de la base de datos para dárselos a la capa de lógica de negocio o al revés. La capa de lógica de negocio es la que realiza todas las funciones necesarias a implementar para el correcto funcionamiento de la aplicación. Realiza las funciones que le pide el usuario recibidas de la capa de presentación utilizando los datos que le proporciona la capa de datos para devolverle el resultado de nuevo a la capa de presentación. La capa de presentación es la que interactúa con el usuario y envía las peticiones que desea el usuario a la capa de lógica de negocio y espera el resultado para mostrárselo al usuario.

En la mayor parte del proyecto hemos mezclado en una la capa de datos y la de lógica de negocio, pero sí que en el caso de las funciones y los servicios de autenticación de *Firebase*, hemos utilizado las tres capas creando un proveedor de servicio de autenticación para que pudiese ser usado en todo momento para que el usuario pueda cerrar sesión cuando desee y para obtener información sobre el usuario que está utilizando la aplicación en ese momento.

# Implementación

En esta sección se explican y detallan los aspectos y pasos más importantes realizados para la implementación del proyecto.

### Preparación del entorno

En primer lugar se ha preparado el entorno en el que se va a programar la aplicación. Para ello se han seguido los diferentes pasos que voy se van a explicar a continuación. Antes de explicar los pasos a seguir, decir que, el sistema operativo usado en todo momento para el desarrollo del proyecto es *Windows*, cambiando entre *Windows 7* y *Windows 10*, siendo los mismos procedimientos en ambas versiones de *Windows*.

Lo primero ha realizar es la instalación de *Ionic 2*. Para ello lo primero que hay que hacer es la instalación de *NodeJS* y *Cordova*. Una vez instalado esto, hemos instalado *Ionic 2*.

Una vez instalado *Ionic 2*, ahora lo que realizamos es la creación de un proyecto de *Ionic 2*, para lo cual, accedemos al directorio en el que se desea crear el proyecto con la consola de comandos y ejecutamos el siguiente comando para crear el proyecto:

*ionic start nombre\_proyecto sidemenu –v2*

Donde en nombre\_proyecto ponemos el nombre que deseamos que tenga nuestro proyecto. La palabra *sidemenu* hace que el proyecto creado tenga generado ya por defecto la estructura para incluir el menú desplegable que hemos visto que deseamos tener en los prototipos de las interfaces.

Ahora tenemos que añadir al proyecto creado las plataformas en las que queremos que funcione nuestra aplicación. Para conseguir esto, accedemos al directorio en el que se encuentra nuestro proyecto. Y ejecutamos el siguiente comando una vez por cada plataforma que queremos añadir a nuestro proyecto:

*ionic platform add nombre\_plataforma*

Poniendo en nombre\_plataforma el nombre de la plataforma que vamos a añadir.

Ya tenemos preparado el entorno de *Ionic 2*, pero tenemos que añadir a este proyecto el sistema de base de datos que vamos a usar que es *Firebase*, para que podamos acceder a nuestros datos almacenados en nuestro proyecto de *Firebase* desde nuestro proyecto de *Ionic 2*.

Lo primero que hacemos es crearnos una cuenta en *Firebase* y crearnos un nuevo proyecto dentro de esta.

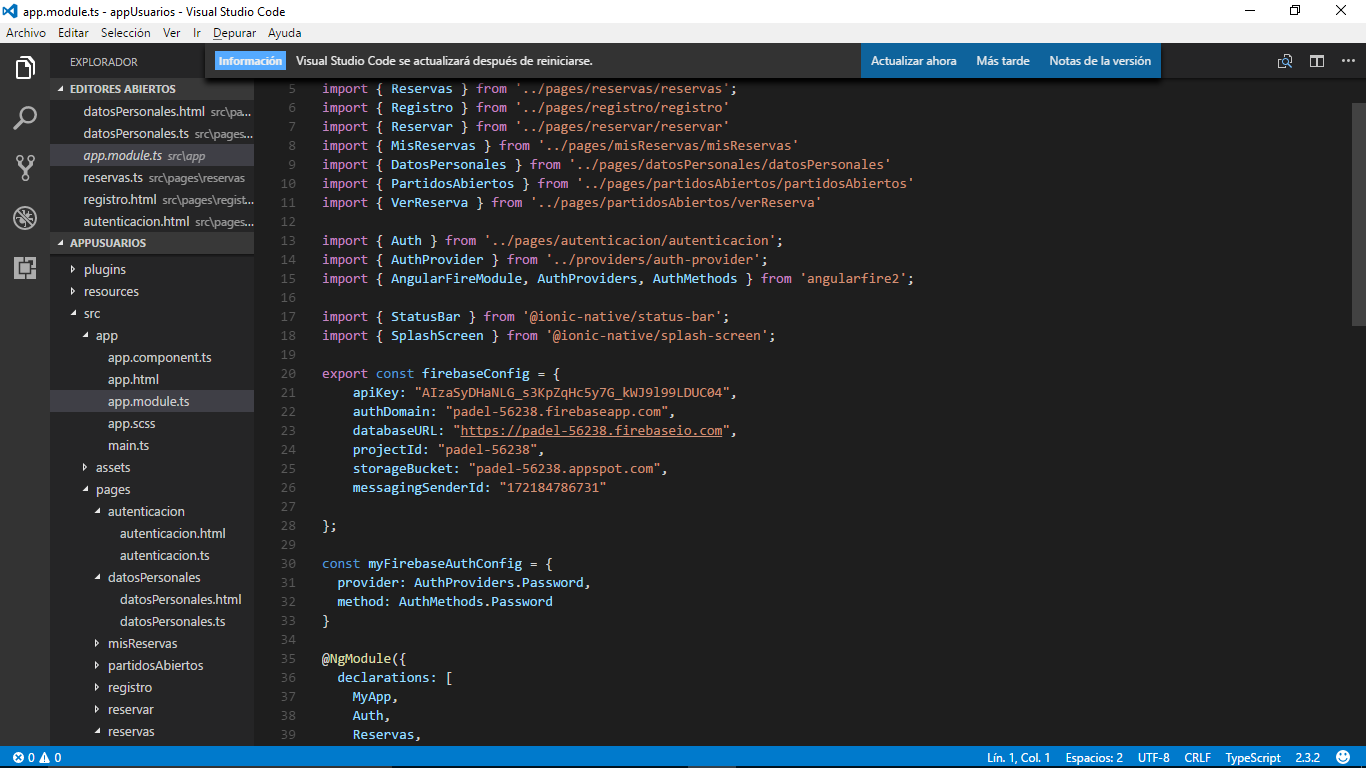
Cuando ya tenemos el proyecto creado en *Firebase*, tenemos que añadir este proyecto a nuestro proyecto en *Ionic 2*. Para conseguir esto, tenemos que acceder de nuevo a nuestro directorio del proyecto de *Ionic* con la consola de comandos de *Windows* y ejecutamos los siguientes comandos:

*npm install @ionic/app-scripts@latest –save-dev*

*npm install firebase angularfire2 –save*

De esta forma, ya podemos importar las dependencias de *Firebase* desde nuestro proyecto. Lo siguiente a hacer es añadir el proyecto particular que hemos creado en *Firebase* a nuestro proyecto en *Ionic*. Para ello obtenemos desde *Firebase* un código que está aplicación nos proporciona para añadir su proyecto a aplicaciones web que es la forma genérica para que funcione en todas las plataformas y añadimos ese código a nuestro proyecto de *Ionic* en el archivo de configuración llamado *app.module.ts*.

El código a añadir es el siguiente en el caso particular de mi proyecto de *Firebase*:

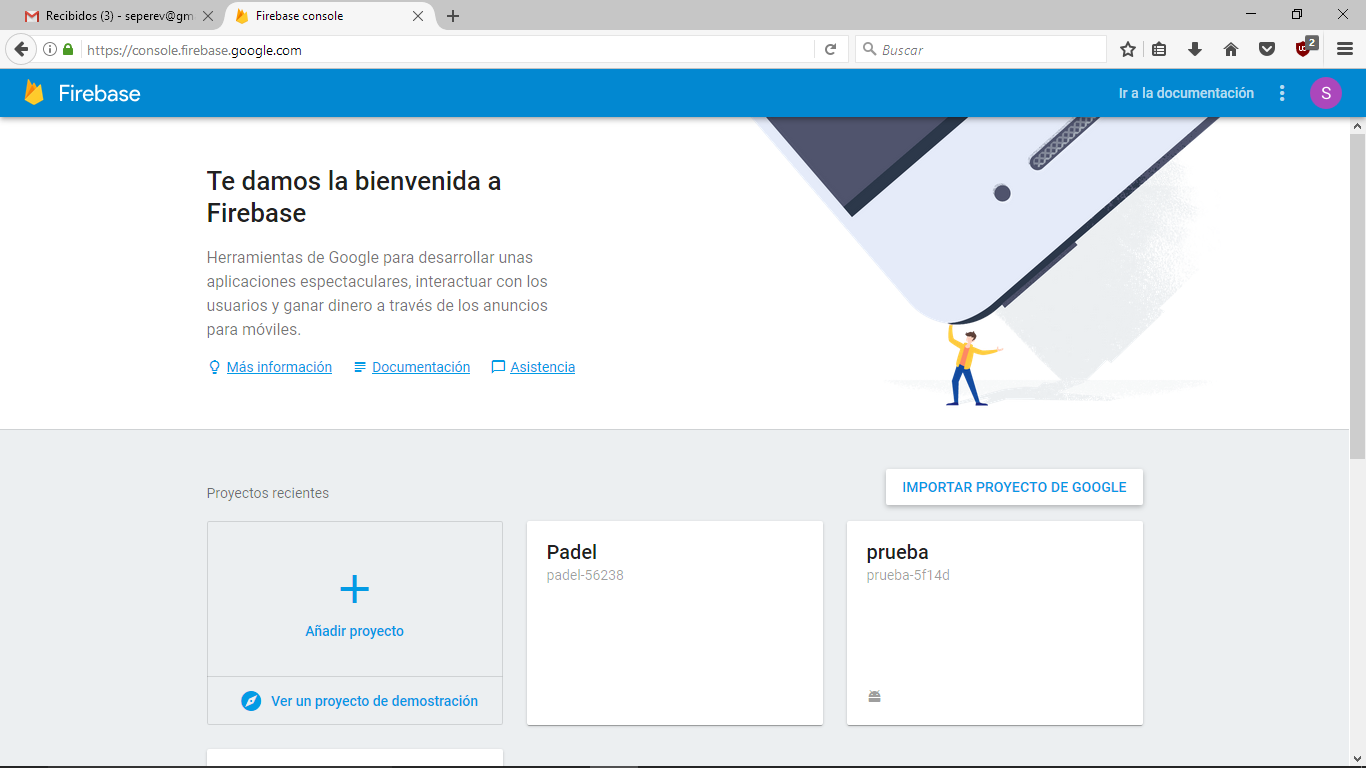


### Preparación de la base de datos

Como se ha dicho anteriormente, la base de datos va a estar almacenada en la aplicación en línea Firebase.

Para crear la base de datos lo que hay que hacer es crearnos una cuenta de correo electrónico de *Google* (*Gmail*) y acceder a *Firebase* desde la página web (ver bibliografía [16]).

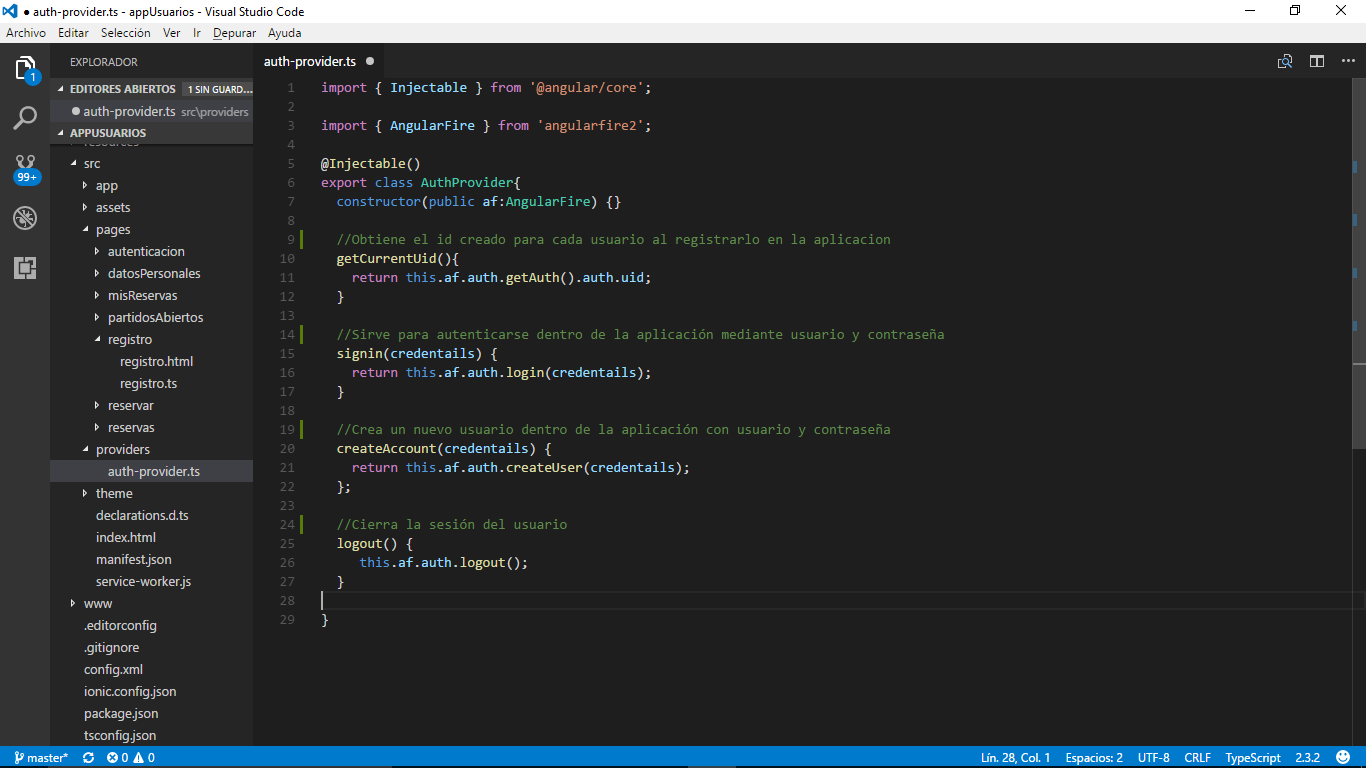
Desde ahí accederemos a nuestra cuenta en la que se almacenarán los proyectos en los que estamos trabajando como podemos ver en la siguiente imagen.



Como podéis ver en nuestra cuenta hay creados dos proyectos diferentes llamados ‘Padel’ y ‘prueba’.

Al acceder a uno de ellos podemos ver los diferentes servicios que ofrece *Firebase* nombrados anteriormente. Ahora tenemos que configurar los servicios que hemos nombrado anteriormente que íbamos a utilizar para nuestro proyecto.

* Authentication: dentro de este servicio podremos elegir de qué forma queremos que los usuarios se autentiquen dentro de nuestra aplicación. En nuestro caso, hemos habilitado la opción para que los usuarios accedan por medio de un correo electrónico y una contraseña. Este correo electrónico y esta contraseña será almacenada en este servicio utilizando un proveedor que nos hemos tenido que crear para usar las funciones que nos da *Firebase* para usar esas funciones durante toda la aplicación.

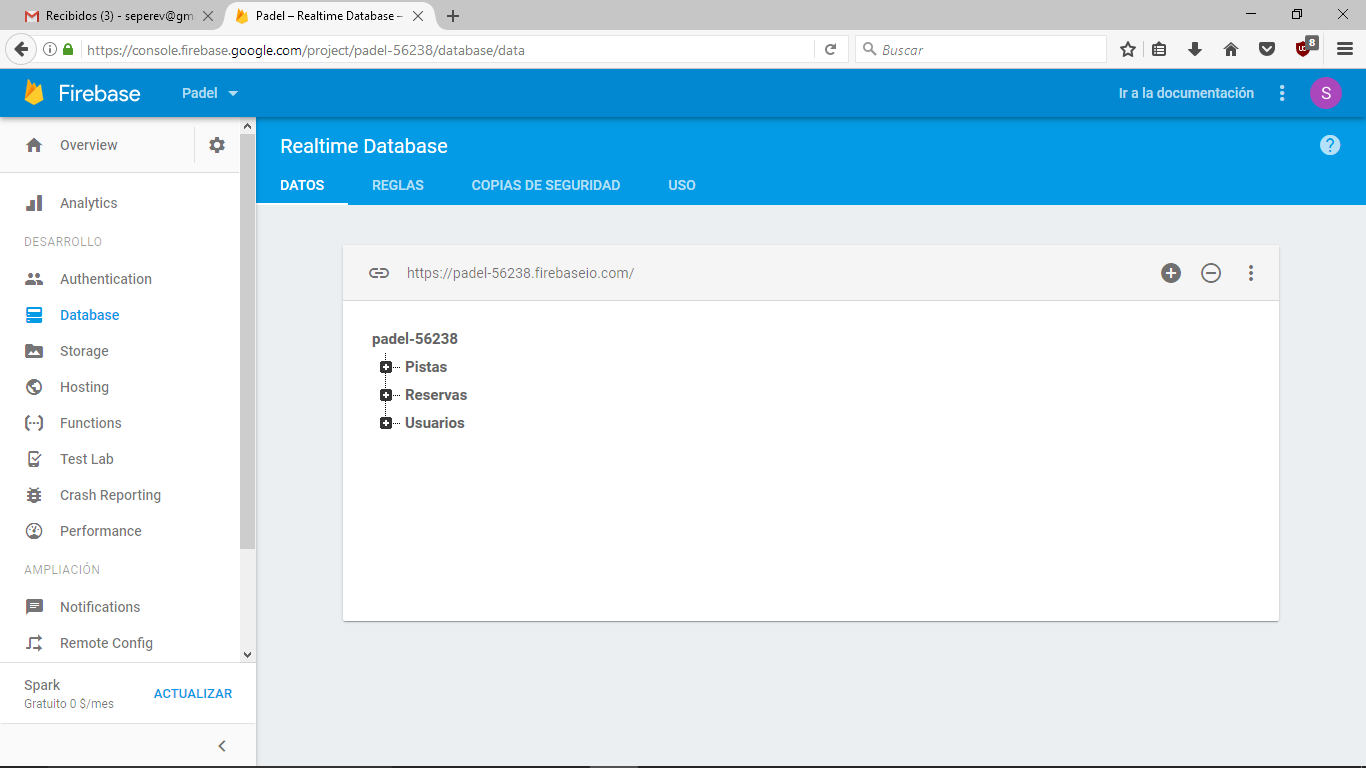


* *Realtime database*: en este servicio es donde realmente se almacenan los datos de la aplicación como los usuarios, las pistas y las reservas cada una con sus datos. Esta estructura de datos se puede crear de dos maneras:
* Se puede crear la estructura manualmente introduciendo las estructuras y subestructuras a mano desde el servicio de la página de *Firebase*.
* Y también se puede ir creando la estructura añadiendo los datos al ir ejecutando la aplicación sin necesidad de que haya generada una estructura anterior. El problema que tiene esta aplicación al dejarte hacer esto es que siempre se tiene que seguir la misma estructura de datos con exactamente los mismos nombres diferenciando entre mayúsculas y minúsculas porque si no puede generar estructuras separadas que pueden dar muchos problemas a la hora de probar el código.

Por todo esto, es por lo que hemos realizado un diseño de la base de datos anteriormente para que cuando tengamos que crear elementos dentro del código de nuestra aplicación, siempre insertemos los mismos datos, con la misma estructura y con los mismos nombres.

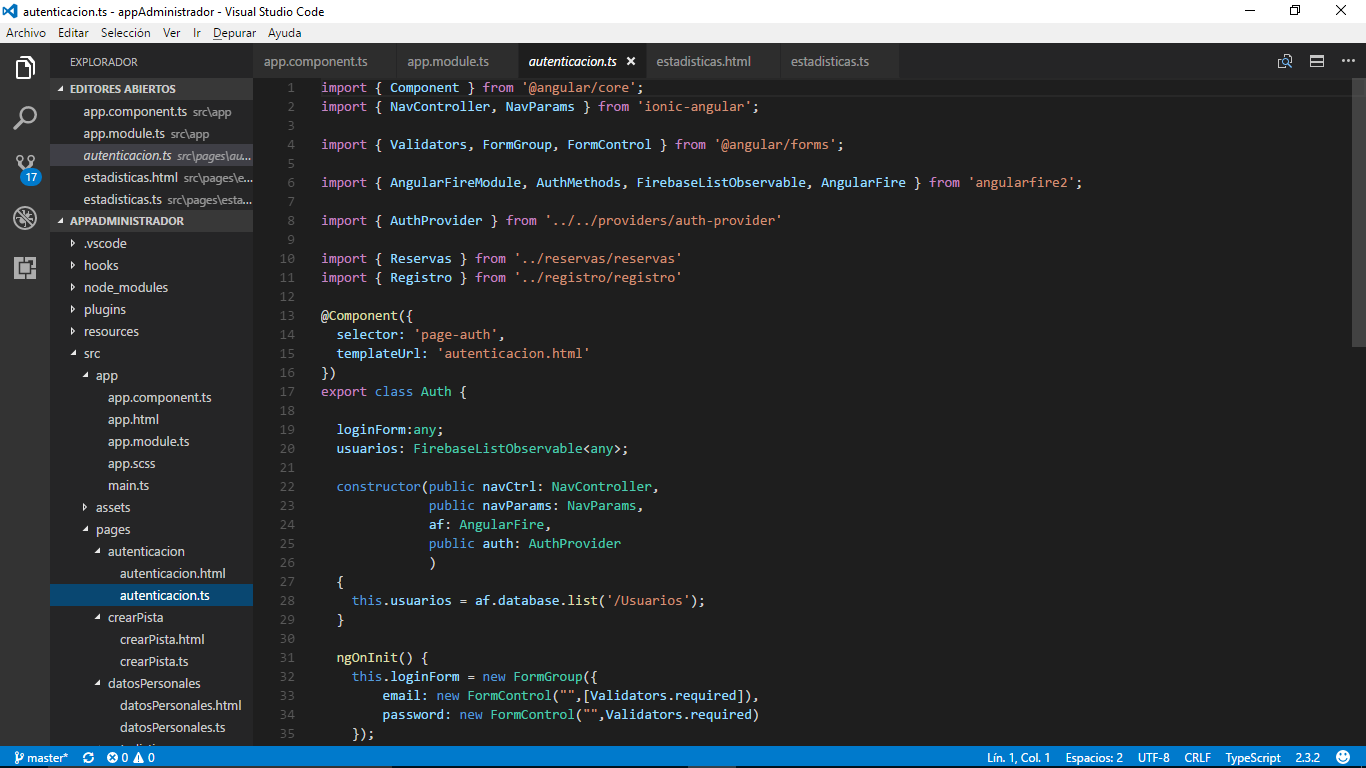
Para sincronizar este servicio con el servicio de autenticación, necesario para conocer en cada momento el usuario que está conectado en ese momento en la aplicación, hemos tenido que almacenar en este servicio en los usuarios un datos llamado ‘*uid’* que es la copia del que genera y almacena el servicio de autenticación cuando crea un usuario nuevo en la aplicación.

Para almacenar los datos dentro de este servicio, lo que hemos realizado primero es crear las tres grandes estructuras de datos manualmente desde la aplicación de *Firebase* que son ‘*Usuarios’, ‘Pistas’* y *‘Reservas’*.



* + 1. Inserción de datos

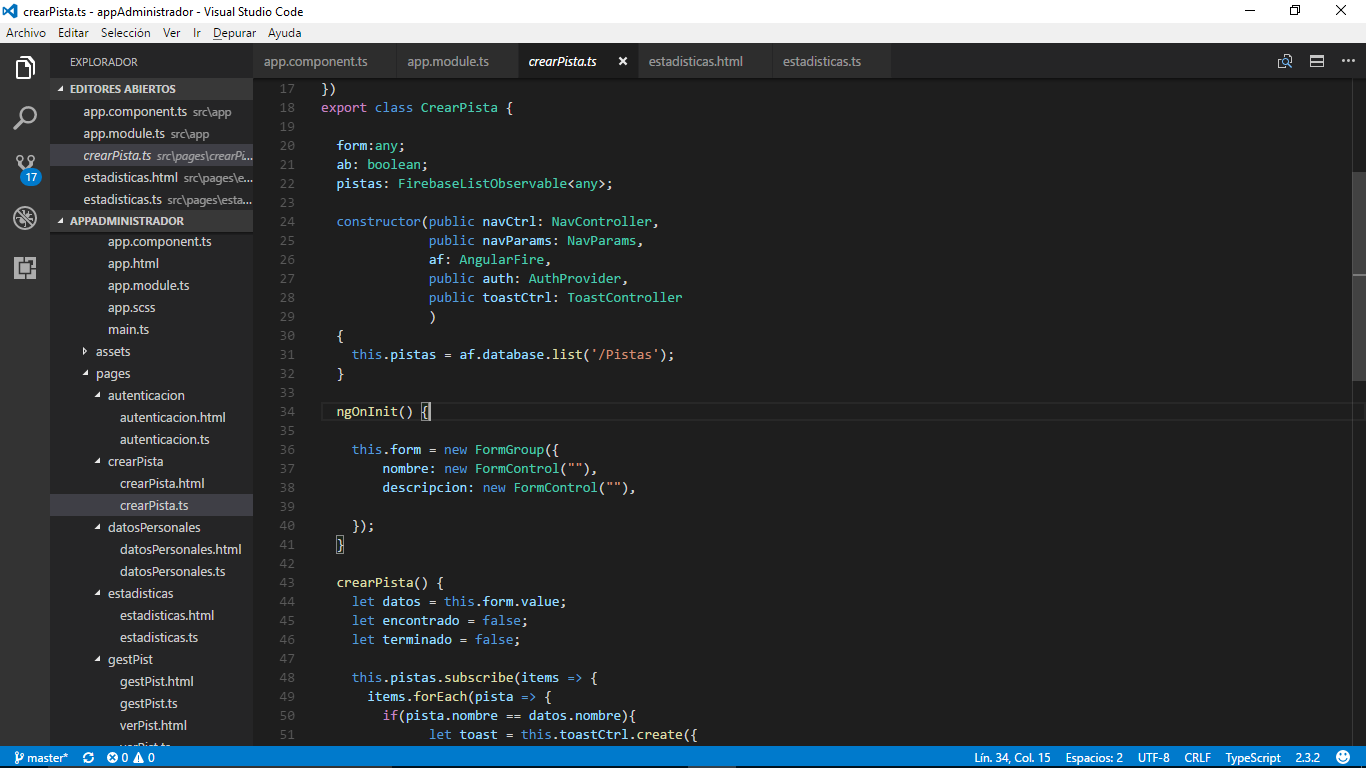
Y ahora, después de haber creado estas tres estructuras he introducido los datos dentro de las mismas mediante código mientras iba creando la aplicación final. Para esto, tenemos que importar el módulo de *AngularFire* en las clases en las que queremos enviar o recibir datos de la base de datos de la siguiente forma:



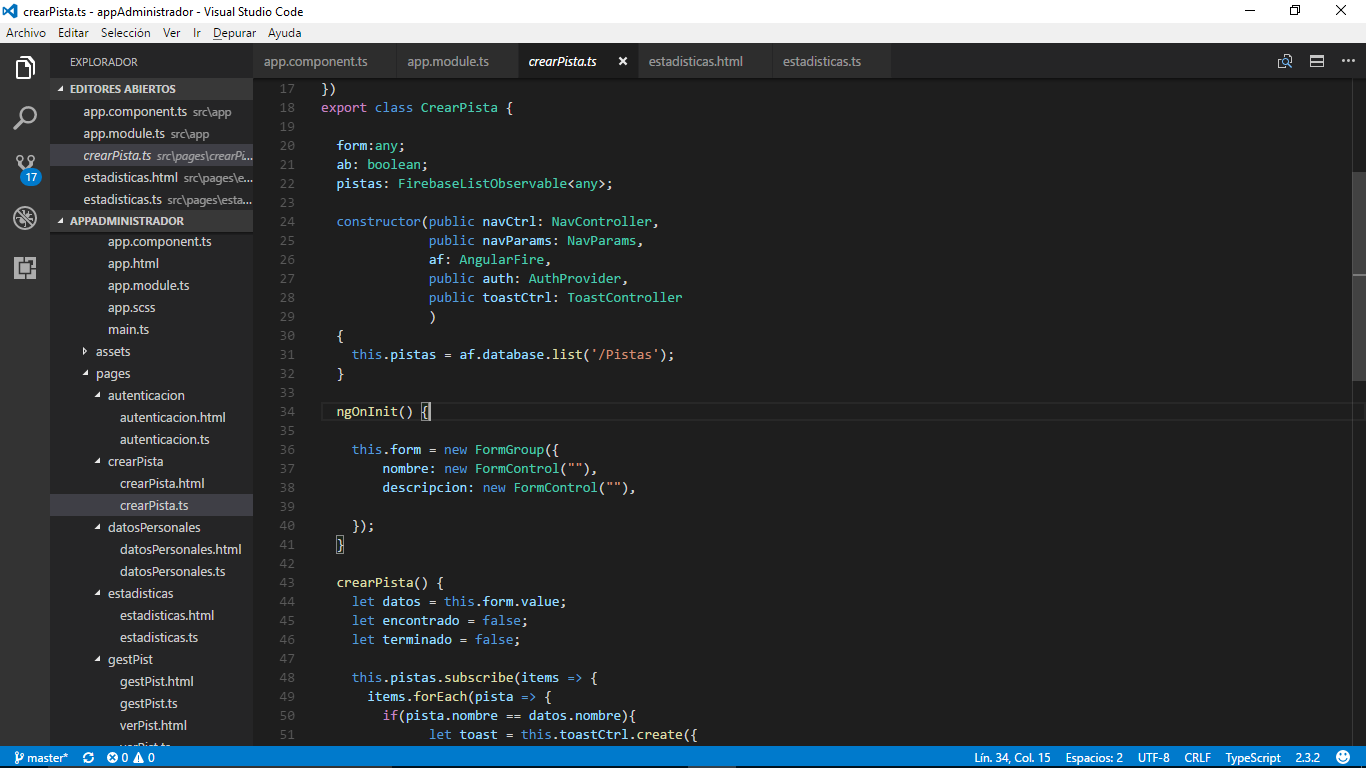
Como vemos en esta línea añadimos diferentes módulos de *AngularFire* necesarios la conexión con la base de datos.

Ahora que tenemos ya importados los módulos necesarios, si queremos crear una nueva pista con sus datos, lo primero que tenemos que hacer es crearnos una referencia a la base de datos, en concreto, a la estructura llamada ‘*Pistas’* que es donde vamos a introducir todas las pistas y se hace de la siguiente forma:

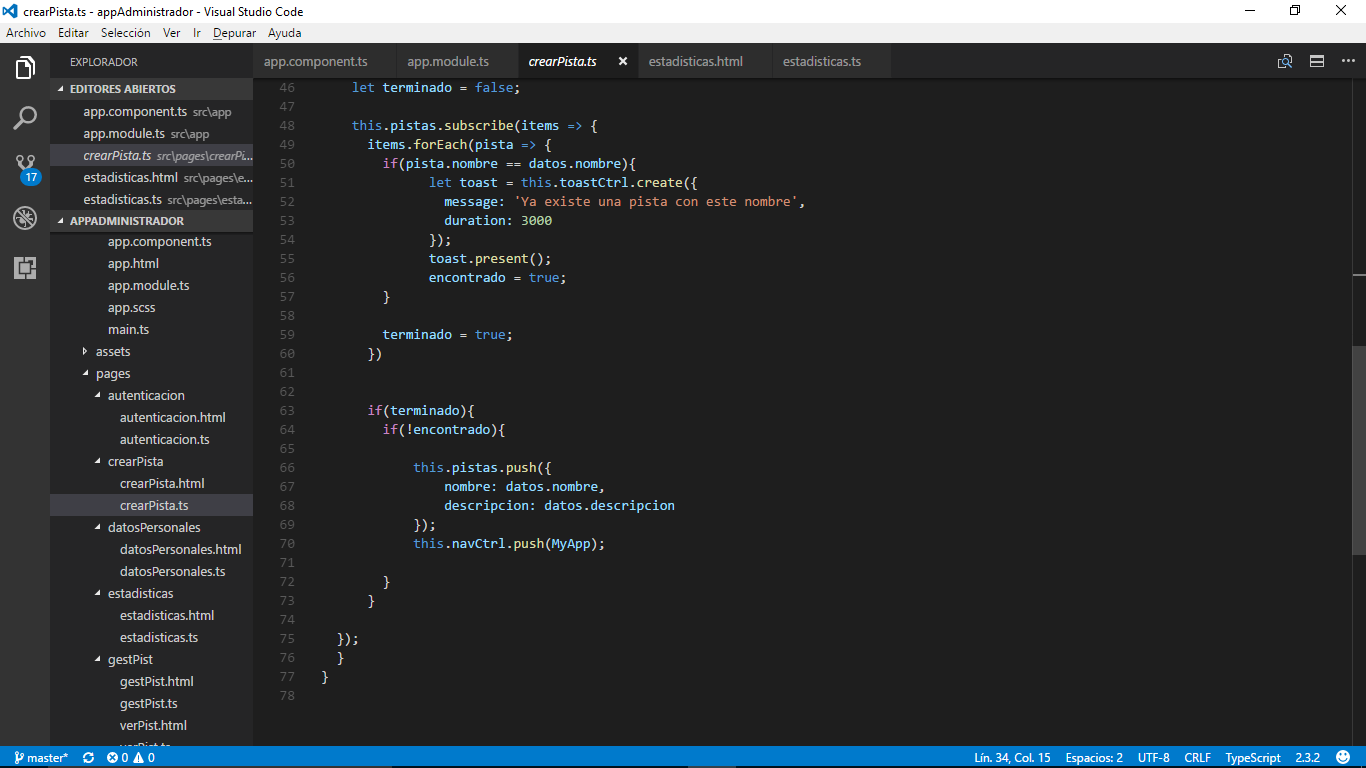
* Primero nos creamos una variable del tipo *FirebaseListObservable<any>* que es la que va a contener la referencia a la base de datos:



* Lo segundo usamos una referencia a *AngularFire* con un método que nos proporciona para conseguir la referencia deseada:

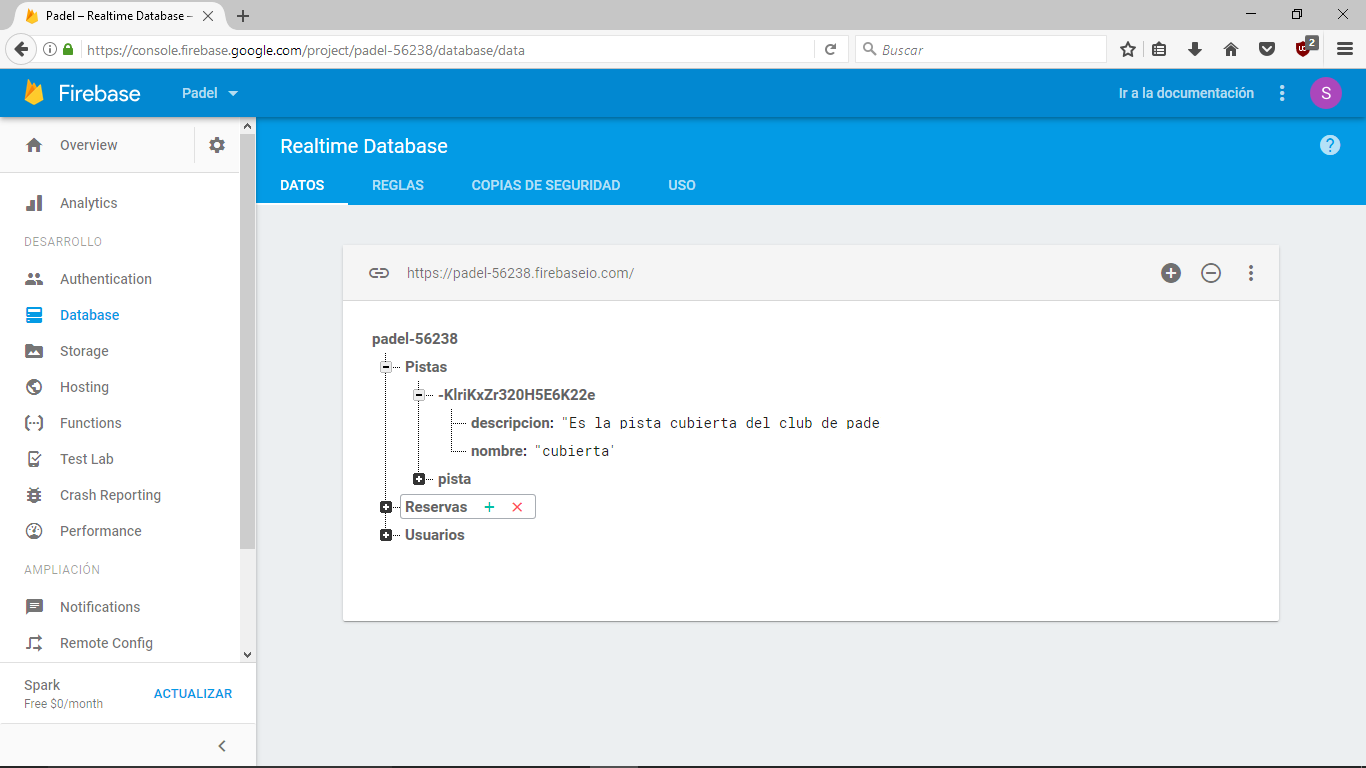


Por último, para añadir la pista que deseamos con los datos que queremos utilizamos la función ‘*push*’ de la referencia de la siguiente forma:



Lo que conseguimos con esto es crear una nueva pista que se generará con un nombre definido por el elemento ‘*datos.nombre’* y una descripción definida por ‘*datos.descripcion’*.

Esto nos genera en la base de datos de *Firebase* lo siguiente:

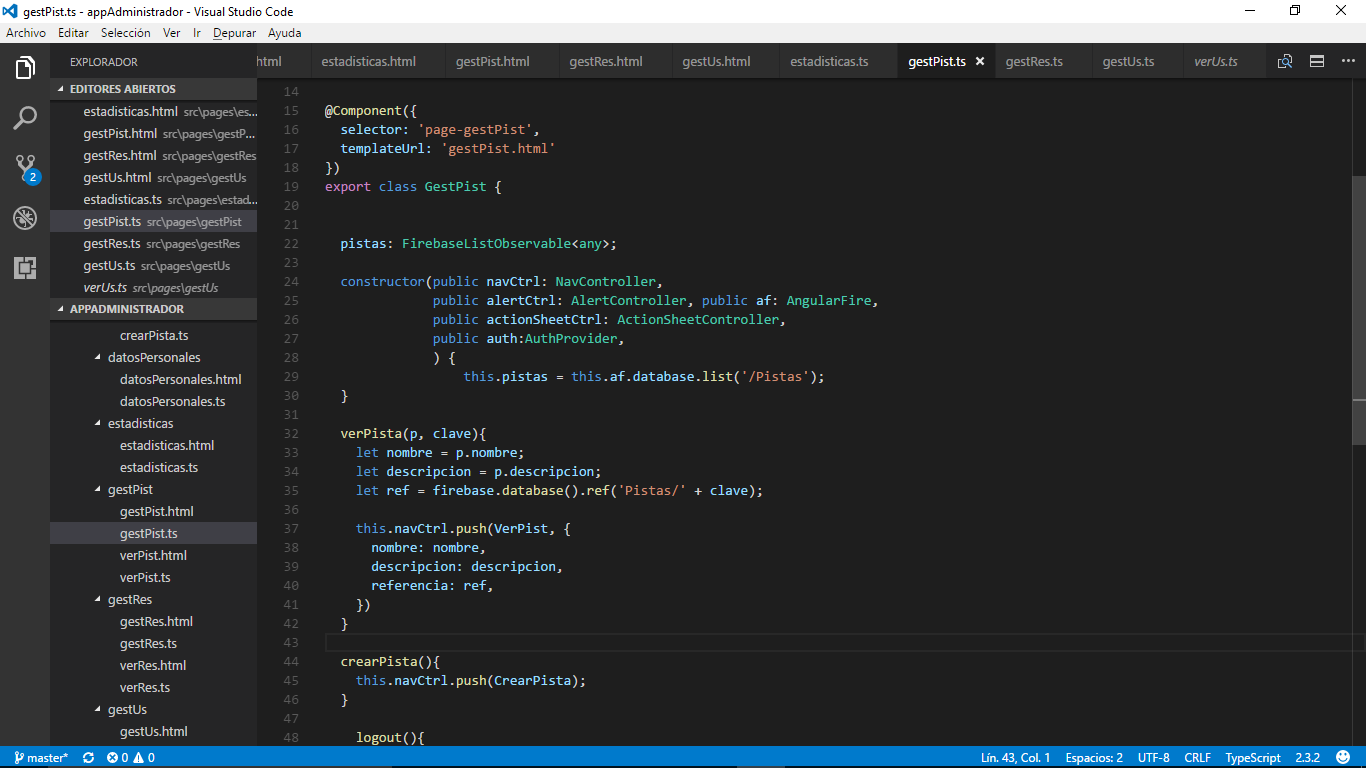


Como vemos, nos genera dentro de la estructura creada anteriormente llamada “Pistas” un nueva pista, que como vemos nos la genera con una clave única que tienen todos los elementos que creamos para esta base de datos y dentro de esa clave único nos genera los valores que hemos introducido que son el nombre de la pista y su descripción con el valor definido por el usuario.

* + 1. Recuperación de los datos

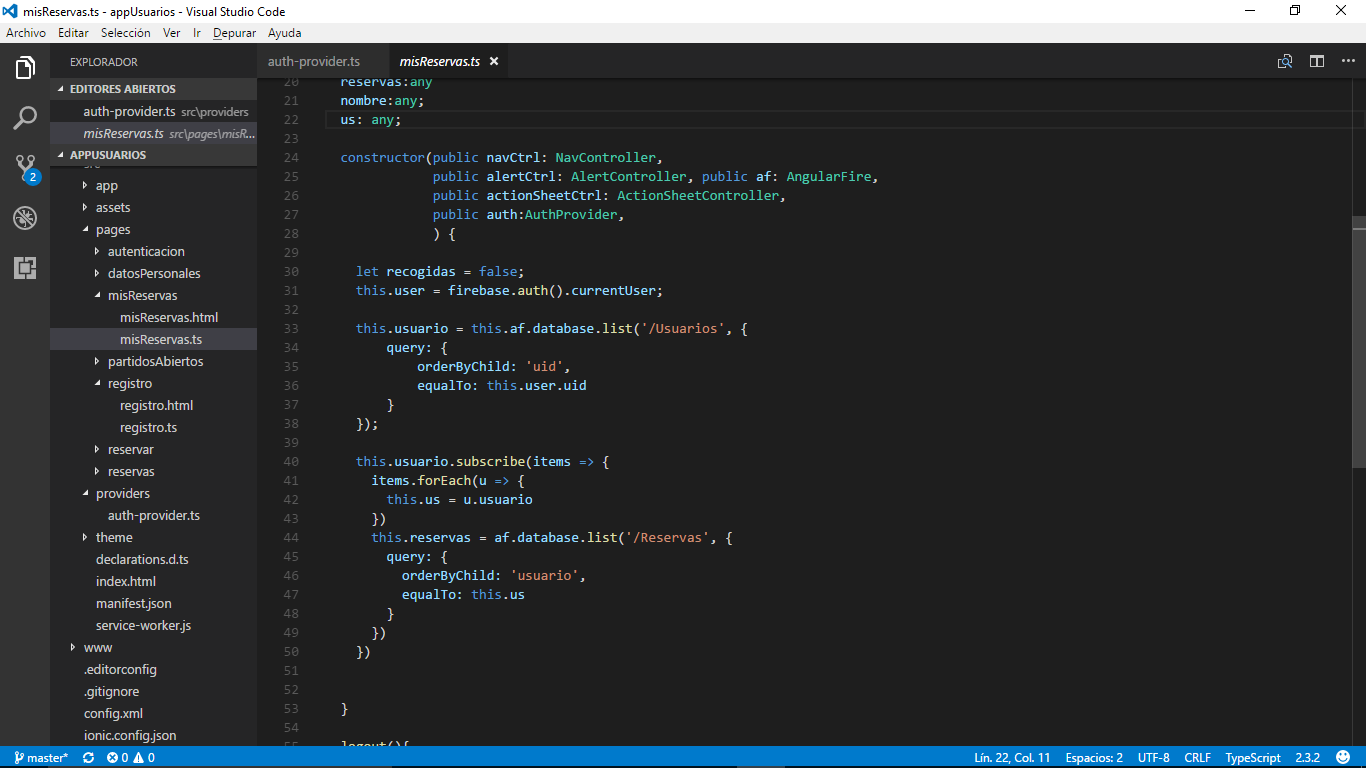
Cuando ya tenemos todos los datos necesarios introducidos en la base de datos, ahora lo que necesitamos es recuperar esos datos. Hay diferentes formas de recuperar esos datos dependiendo de para que vayas a utilizarlos.

Si lo que deseamos es recuperar todos los elementos de una estructura, como por ejemplo, todas las pistas que hay almacenadas en la base de datos, lo que hacemos es utilizar *AngularFire* para devolver a una variable *FirebaseListObservable<any>* todas los elementos que hay dentro de la referencia de la base de datos llamada “Pistas” de la siguiente forma:



De esta forma tendremos en la variable pistas todos los elementos que haya dentro de la referencia “/Pistas”.

Pero, si por el contrario, tan solo queremos recoger los elementos que cumplan una condición en concreto dentro de la base de datos, como por ejemplo, coger las reservas que hayan sido reservadas por el usuario que está ahora mismo utilizando la aplicación, tenemos que usar la función “query” de la siguiente forma:



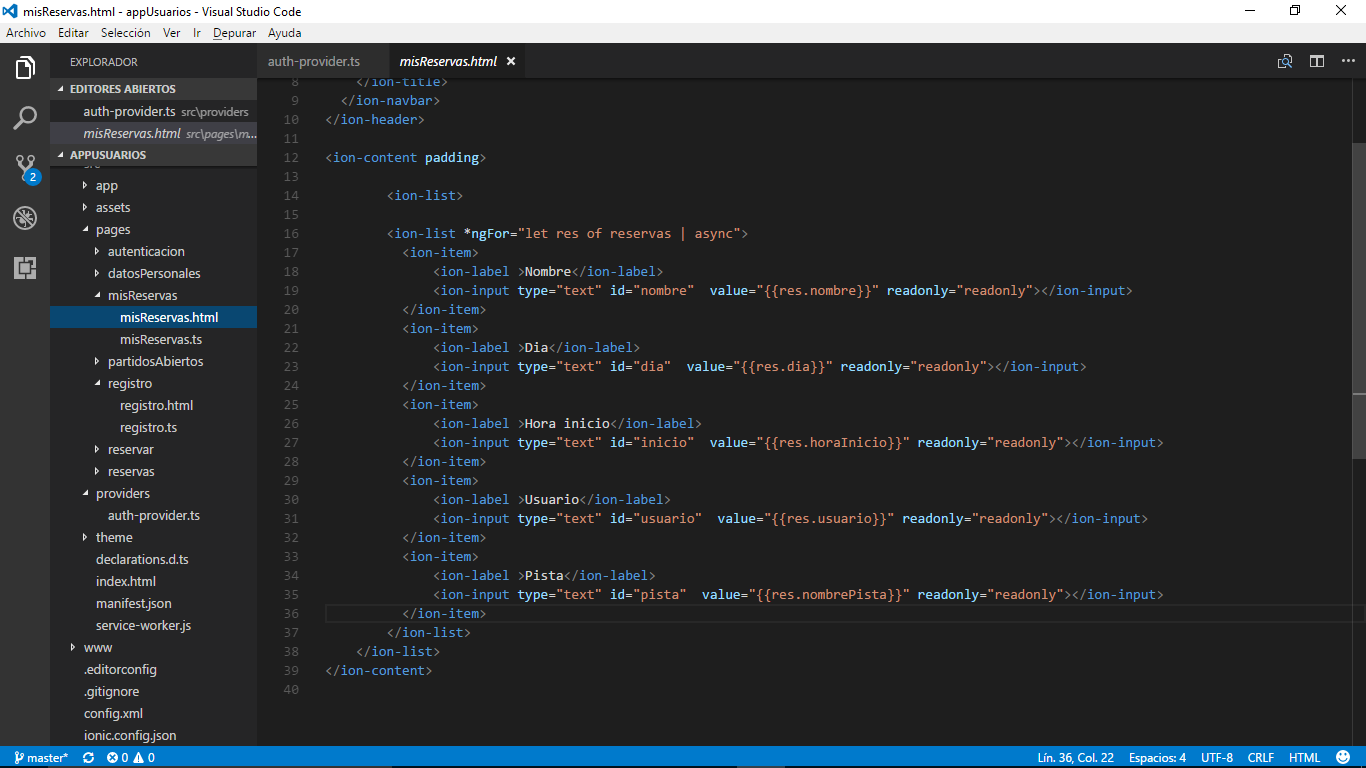
Como vemos, realizamos el mismo paso de antes, pero ahora utilizamos la función “query”. Como ya hemos dicho anteriormente, las “Reservas” van a tener un atributo llamado “usuario” donde se almacena el nombre del usuario que ha realizado la reserva que es único. Entonces lo que hacemos es ordenar las reservas por el atributo “usuario” y luego coger los que sean igual al nombre de usuario que está actualmente utilizando la aplicación.

* + 1. Manejo de los datos

Ahora que ya tenemos los datos recuperados de la base de datos, ahora los podemos usar para diferentes resultados. En particular, para las dos opciones que hemos utilizado estos datos son:

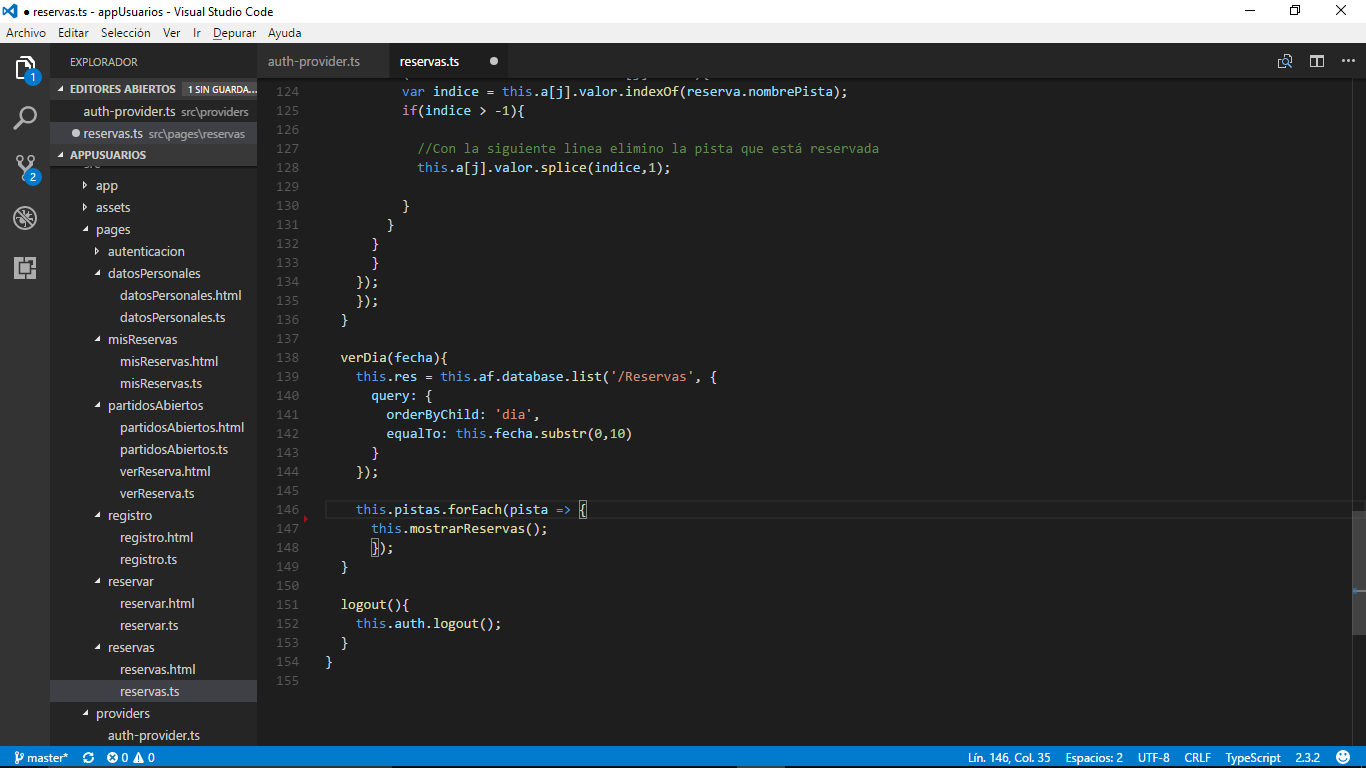
* Para representar esos datos en una vista. Una vez que tenemos, por ejemplo, la lista de las reservas que ha realizado el usuario, podemos mostrar en la vista una lista con todas las reservas del usuario y sus datos más significativos.

Para realizar esto, utilizamos la característica de *AngularJS* que nos permite el paso bidireccional de datos entre la vista y el controlador con tan solo crear en la clase del controlador una variable pública y por lo tanto para mostrarlos en la vista se haría:



Como vemos recorre con la función de *AngularJS* la variable reservas, que tiene que estar creada como variable pública en la clase del controlador. Cada vez que recorre una reserva, como vemos, recoge los datos deseados poniendo los valores entre doble corchetes.

* La otra utilidad que le hemos dado a los datos recuperados de la base de datos es para manejarlos en los controladores para otras utilidades. Para esto, tenemos que utilizar el método “*forEach*” de las variables *“FirebaselistObservable”* para recorrer todos los elementos de la variable del siguiente modo:



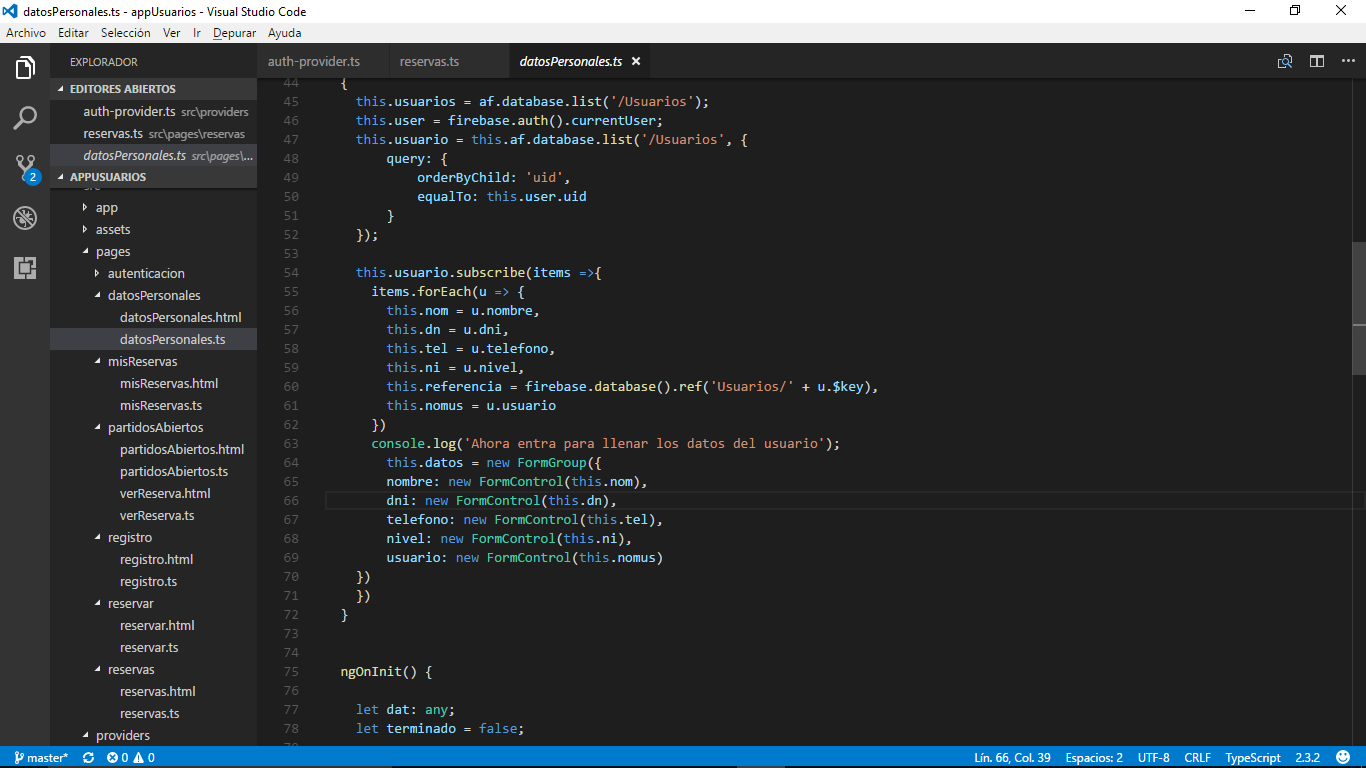
Aquí lo que hacemos es por cada pista que hay almacenada en la variable pistas, se llama al método “this.mostrarReservas()*”*.

* + 1. Modificación de los datos

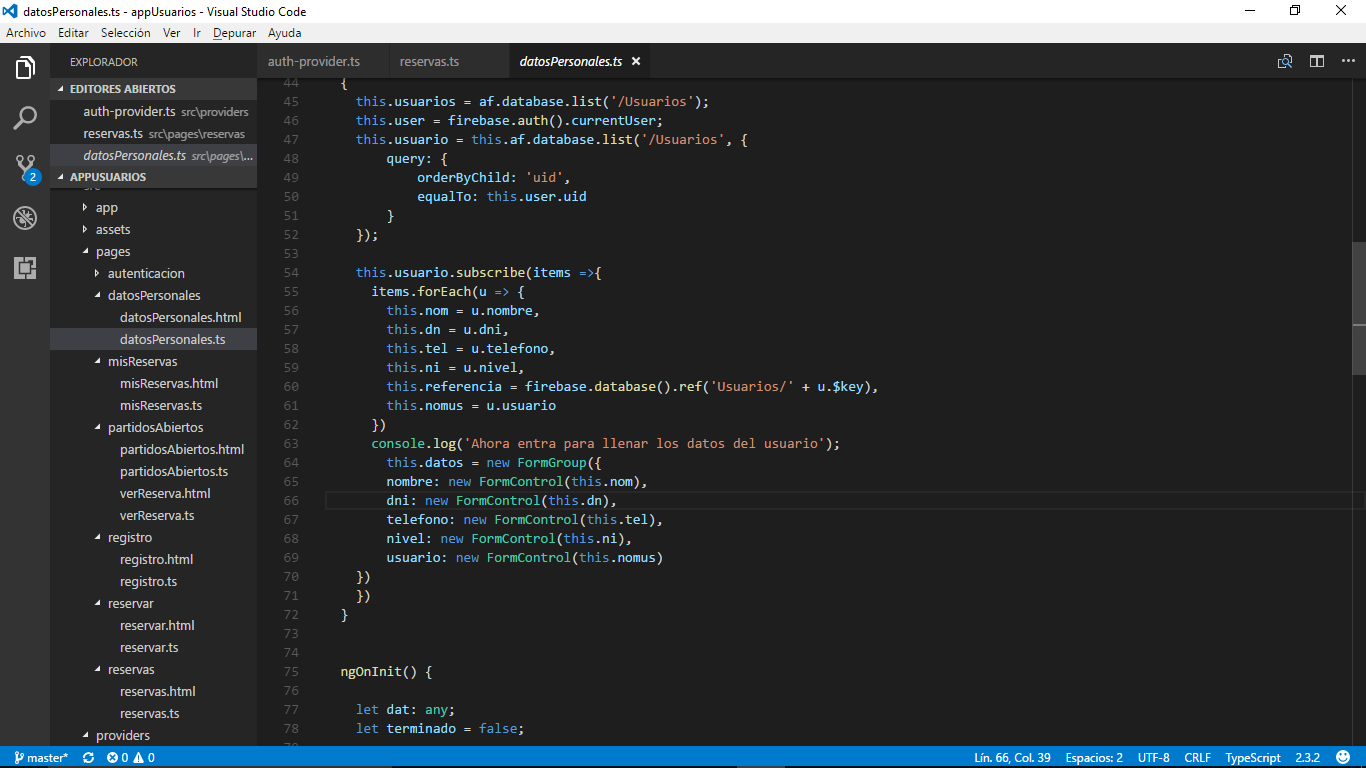
Ya hemos explicado cómo crear datos en la base de datos, cómo recupéralos y cómo manejarlos; ahora vamos a explicar cómo modificar los datos de la base de datos, como por ejemplo, para modificar los datos personales de un usuario.

Para ello lo primero que tenemos que hacer es obtener la referencia al elemento que hay generado en la base de datos dentro de la estructura “Usuarios” que corresponde con el usuario que está utilizando la aplicación que es el que va a modificar sus datos personales.

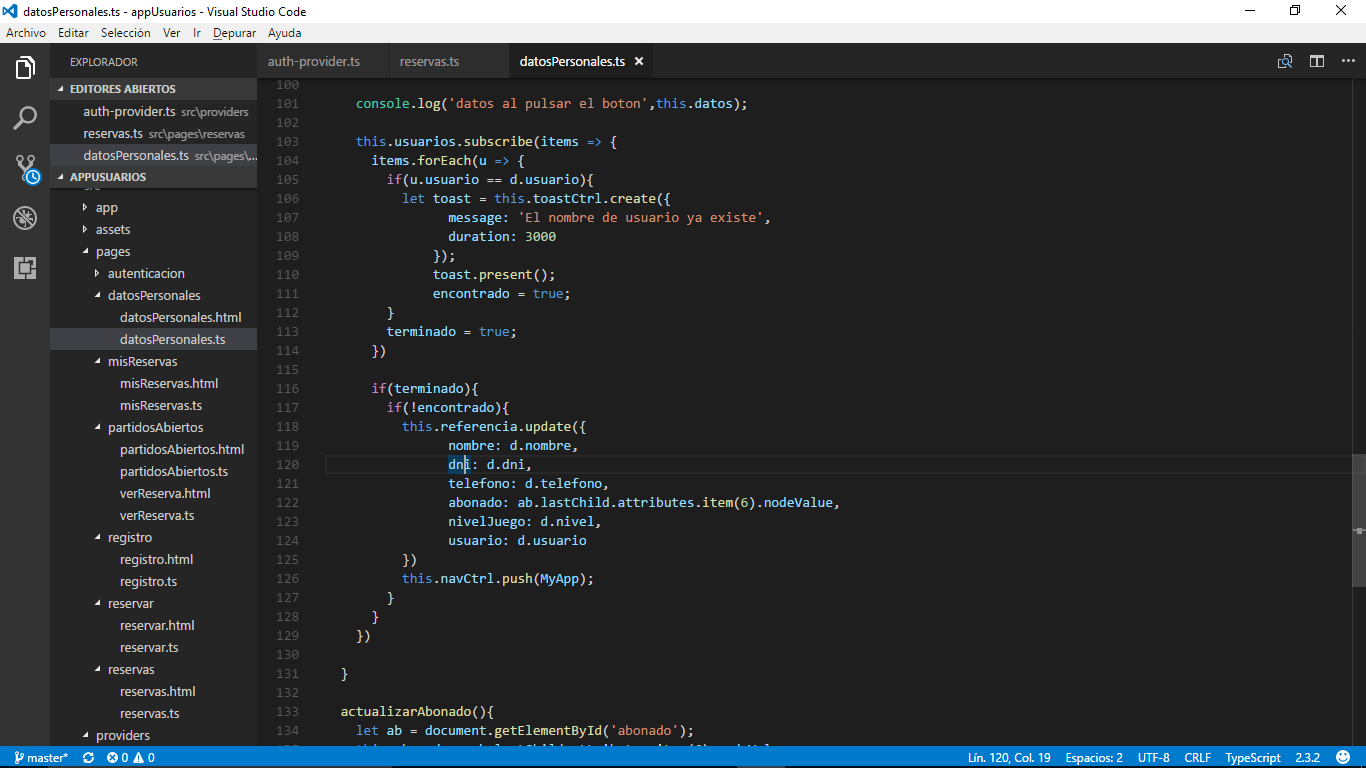
Primero tenemos que recorrer todos los usuarios de la base de datos y obtener el usuario que está actualmente conectado buscándolo por el atributo “uid” del usuario:



Cuando ya tenemos el usuario que queremos modificar sus datos, podemos obtener la clave que hemos explicado antes con la que se generan todos los elementos dentro de *Firebase* con la que podemos identificar cada uno de los elementos utilizando el valor “$key” del elemento que almacena el usuario, y con ese elemento podemos obtener la referencia en concreto al usuario específico que queremos modificar:



Teniendo la referencia al elemento en concreto de la base de datos, ahora podemos modificar los atributos que deseemos utilizando el método “update” del siguiente modo:



Con el siguiente código lo que hacemos es modificar los atributos que hay en la base de datos con el nombre igual a la parte izquierda de los dos puntos poniéndoles el valor que hay a la derecha de los dos puntos. En caso de que no haya un atributo con el mismo nombre al que aparece dentro del “update” este atributo es creado, por esto, hay que tener mucho cuidado y poner el nombre del atributo exactamente igual al nombre del atributo que hay en la base de datos.

### Principales problemas encontrados

Durante la realización del código hemos tenido muchos problemas con respecto a la implementación, casi todos derivados de ser primerizos en la tecnología y del desconocimiento del lenguaje y de la arquitectura a seguir en el proyecto. Nunca antes habíamos utilizado ningún lenguaje usado en la implementación del proyecto a excepción de *HTML* y *CSS* que son los lenguajes que menos he utilizado en toda la implementación debido a su pequeño papel dentro de la aplicación puesto que *HTML* no es usado mucho debido a que *Ionic* proporciona sus propios tags para la utilización de sus propios elementos dentro de las vistas y *CSS* tampoco ha sido muy usado debido a que es una aplicación con una interfaz muy sencilla e intuitiva en la que no se han introducido muchos elementos de estilo y diseño.

El principal problema que nos hemos encontrado durante toda la implementación es el hecho de que la ejecución del código del programa se realiza de forma asíncrona. Esto quiere decir, que el código del programa no se ejecuta de forma secuencial y ordenada, si no que, no se sabe en qué orden se van a ejecutar las líneas de código, porque las funciones asíncronas se ejecutan en segundo plano, mientras que, el programa no espera a que la función devuelva el resultado para seguir ejecutando las siguientes líneas de código.

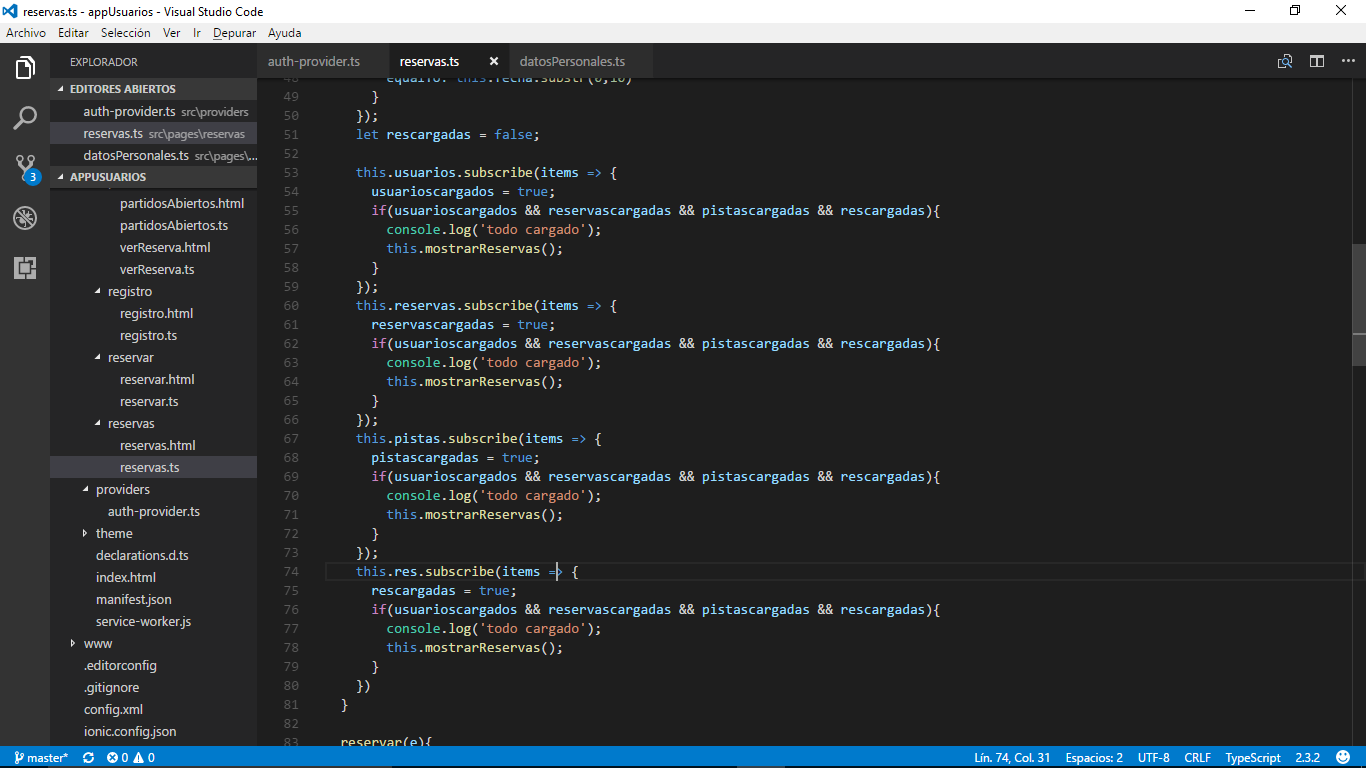
Esta forma de ejecución sirve para mejorar el rendimiento de la aplicación puesto que el programa sigue ejecutando líneas de código sin esperar a las funciones, las cuales, se siguen ejecutando en segundo plano y no ralentizan la ejecución del programa.

Este aspecto nos ha creado un quebradero de cabeza durante toda la implementación de la aplicación puesto que al no controlar la ejecución del código y al ejecutarse en un orden aleatorio, muchas de las partes del código nos daban error puesto que muchas veces ejecutábamos una función que necesitábamos para que nos devolviera un valor que tratábamos más adelante, pero al tratar ese valor, el código daba error puesto que tratábamos el valor antes de que la función terminase de ejecutarse y por lo tanto el valor era indefinido.

Uno de los mayores problemas con este tema ha sido al recoger datos de la base de datos de *Firebase* desde los controladores e intentar tratar esos datos desde el mismo controlador, puesto que, la ejecución no esperaba a que la función de recuperación de los datos de la base de datos terminase de devolverme los datos necesarios, como por ejemplo, en la interfaz principal de las aplicaciones para administrador y usuarios donde teníamos que mostrar las pistas en las diferentes horas del día, en un día determinado, siempre y cuando no haya una reserva de esa pista realizada a esa hora y ese día. Esto nos llevo mucho tiempo solventarlo puesto que intentamos ir depurando el código mostrando los resultados del código en el log de la ejecución del programa, pero lógicamente, tampoco controlábamos cuando se ejecutaba el código de envío de mensajes al log, entonces algunas veces según el log, el código estaba bien escrito, pero el código no hacía lo deseado.

Al final conseguimos solucionar este problema utilizando la función subscribe en cada llamada a la base de datos que hace que la ejecución del código espere a la respuesta de esta función y no continúe con la ejecución del resto del código.

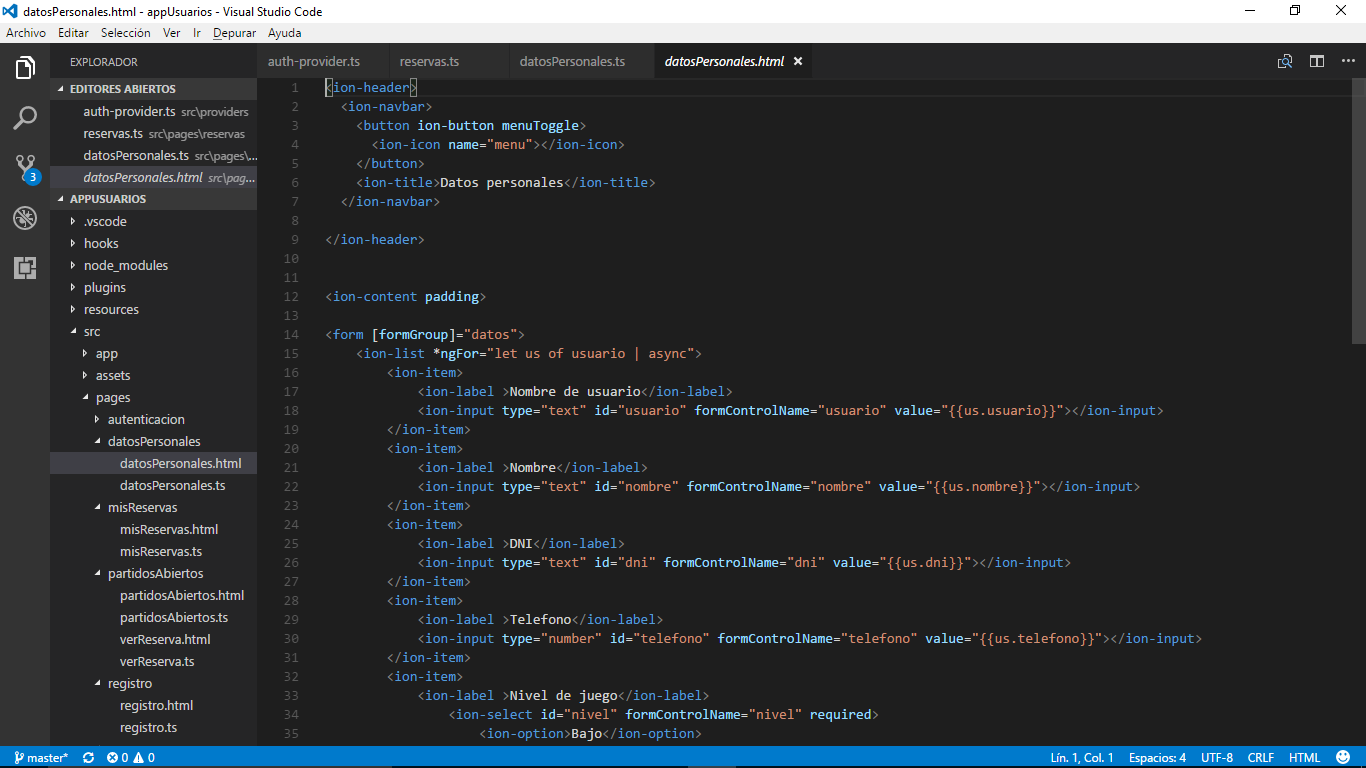
Esto solventaba el caso de recoger datos de la base de datos una sola vez, pero en el controlador de la vista principal de la aplicación, necesitábamos hacer más de una llamada a la base de datos para recoger diferentes datos, por lo tanto, no nos servía sólo con esta función. Lo que tuvimos que hacer es añadir un “*booleano*” por cada llamada a la base de datos para que en cada llamada a la base de datos se pusiera su variable a verdadera y así ejecutar la función que deseábamos solo cuando todas estas variables estuvieran a verdadero de la siguiente forma:



Como vemos en este caso, hemos tenido que recuperar usuarios, pistas y dos veces datos sobre reservas. Dentro de cada función “*subscribe*” ponemos la variable correspondiente a verdadera y además comprobamos si todas las variables están a verdaderas para que en el momento que estén todas a verdaderas esté en el “*subscribe”* que este, se ejecute. La llamada a la función la hemos tenido que poner dentro de todos porque si lo poníamos fuera de ellos, teníamos el problema inicial de que se ejecutase antes de que todos los datos estuvieran cargados.

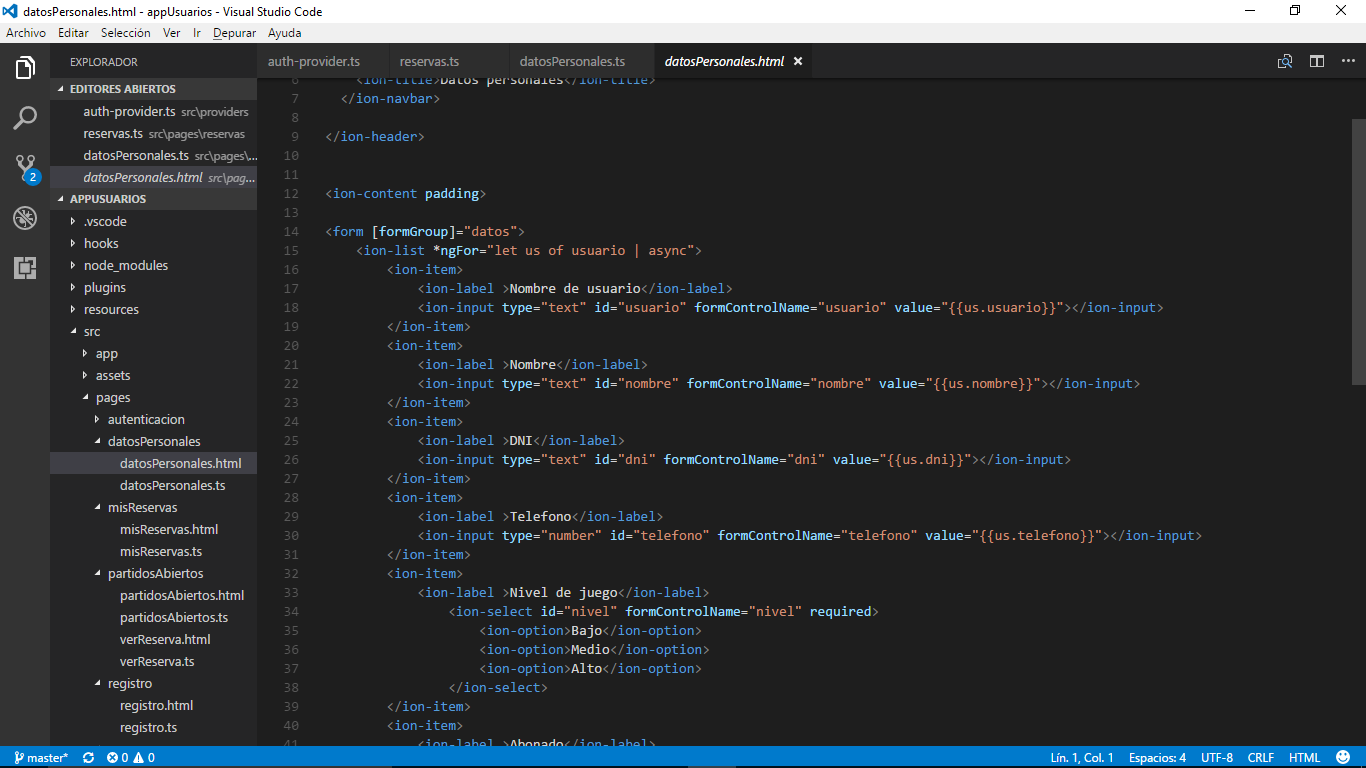
Otro de los mayores problemas que hemos tenido en este aspecto es al querer mostrar los elementos recogidos en un controlador en su vista correspondiente. Como ya hemos explicado anteriormente esto se hace gracias a *AngularJS* que nos permite el paso bidireccional de información de datos entre la vista y el controlador y a que nos da funciones para recorrer listas como “*\*NgFor”*.

Pero, al utilizar está función para mostrar los datos en la vista, al principio no nos aparecía ningún dato, hasta que descubrimos, que el fallo estaba otra vez en la ejecución asíncrona y por lo tanto la vista intentaba mostrar datos sin estar ya cargados en la variable que intentaba recorrer y por lo tanto la lista estaba vacía y no mostraba ningún resultado. Esto lo conseguimos solucionar poniendo en la función “*NgFor”* al final el valor “*async”* para que la vista esperase a que los datos estuvieran cargados por completo en la variable que la vista iba a recorrer. El resultado es:



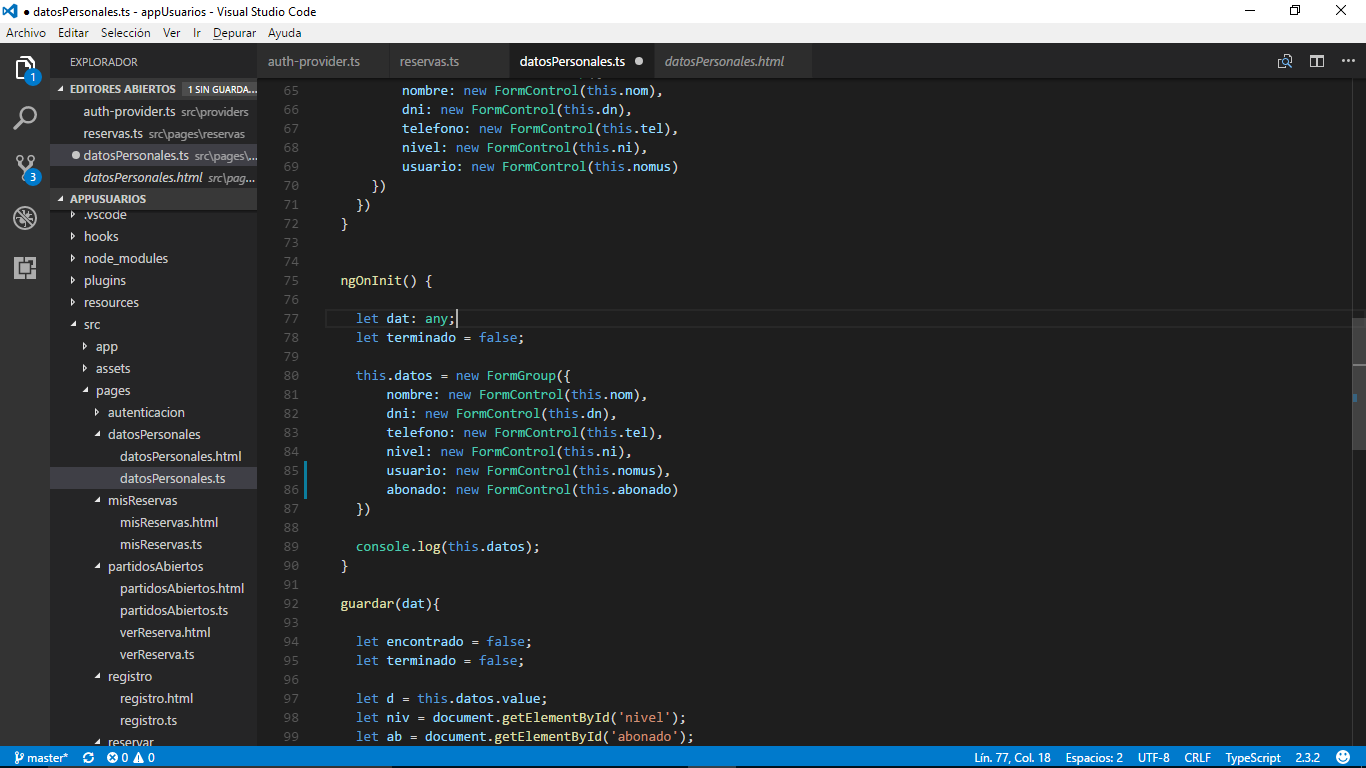
También hemos tenido bastantes problemas con la interfaz de modificación de los datos personales de los usuarios. Esto problema, ha sido debido a que para la creación de formularios he utilizado un “<form>” en la vista, pero con la peculiaridad de añadirle que sea un “FormGroup” para que sea más fácil el paso de datos entre la vista y el controlador.

Por lo tanto, para usar este elemento, en la vista, a cada elemento que deseas añadir al formulario, hay que añadirle un atributo llamado “formControlName” como el siguiente:



Aquí vemos la estructura de cada elemento que aparece en el formulario que como vemos es un “<ion-item>” que siempre tiene que ir dentro de un “<ion-list>” y que dentro tiene un texto y la entrada que va a recoger el formulario además de como vemos, el valor que va a tener que viene definido por el controlador.

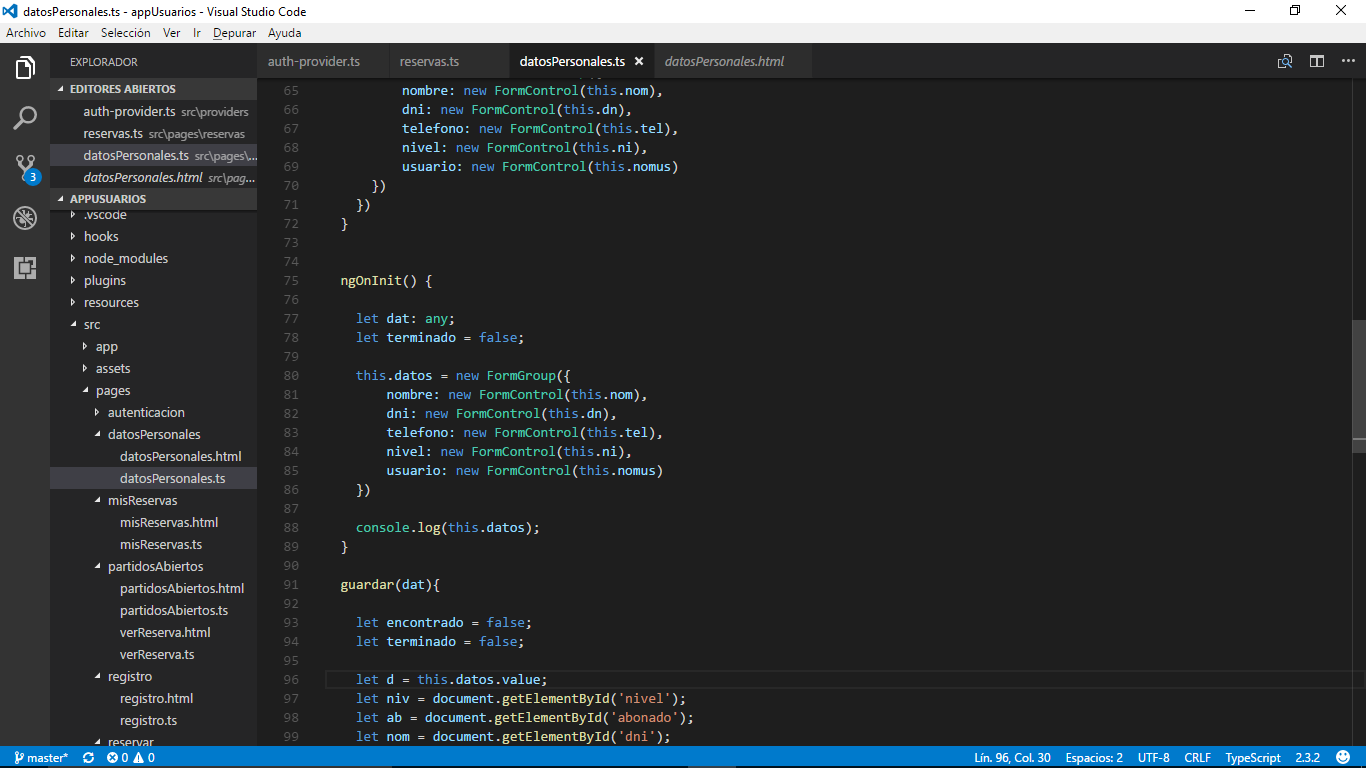
Cuando ya hemos añadido todos los elementos en la vista con su nombre asignado en el formulario, tenemos que crear una variable del tipo “FormGroup” que va a contener todos los elementos del formulario de la siguiente manera:



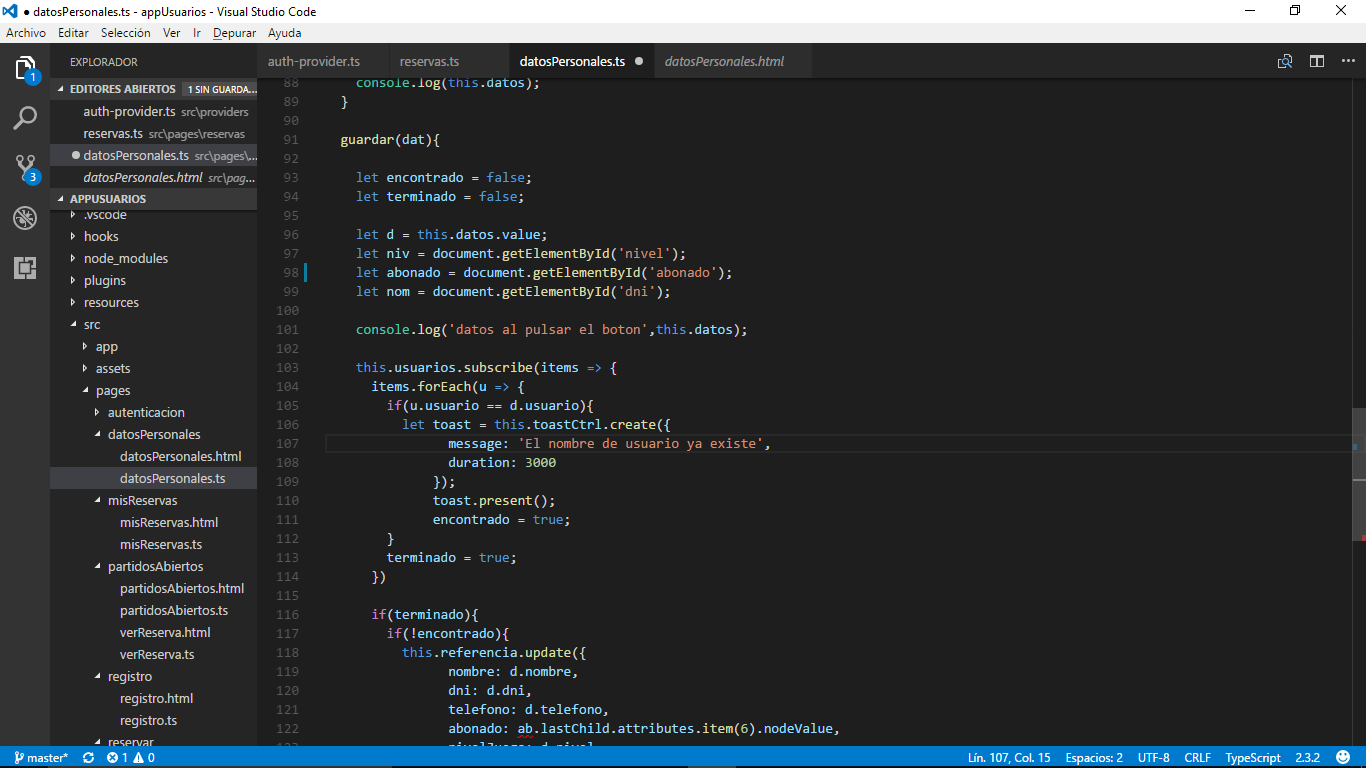
Y utilizando la función “value()” de esa variable, obtenemos otra variable con todos los datos obtenidos de los valores que hay en la vista. Esto sirve, para que cuando el usuario le dé al botón de “Guardar”, podamos obtener los valores que ha introducido el usuario.

Hasta aquí todo funcionaba bien, pero el problema viene cuando hemos introducido un elemento en el formulario que no era una entrada de tipo texto ni de tipo numérico. Hemos introducido como podemos ver en los prototipos de las interfaces un elemento que se llama “toggle” para que el usuario solo tuviera que pinchar sobre él para cambiar el valor de verdadero a falso. Esto lo hemos utilizado, inicialmente, para que el usuario indicase si quería notificaciones y si es abonado al club de pádel o no, pero como ya hemos dicho, las notificaciones no se han llevado a cabo y por lo tanto, finalmente, solo aparece este elemento para que el usuario indique si es abonado o no.

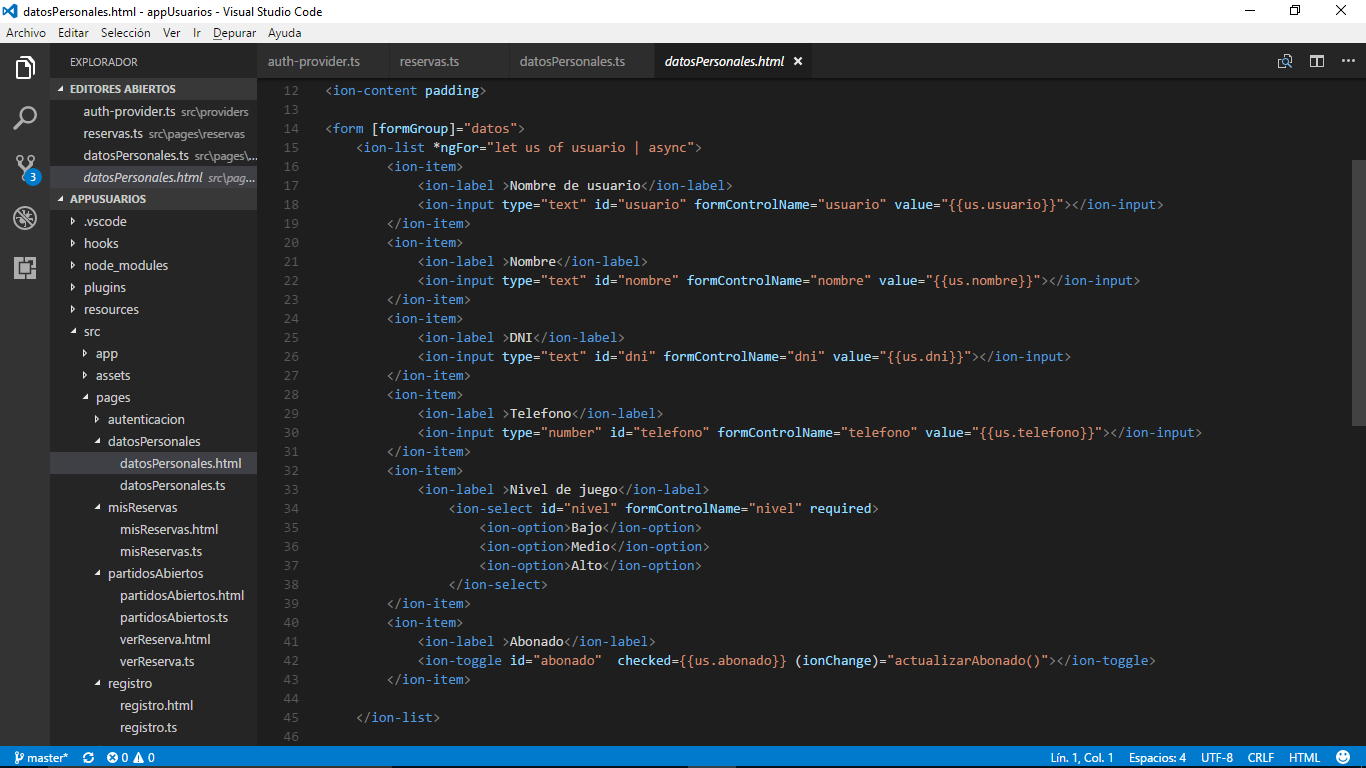
Este elemento, nos ha dado muchos problemas porque al intentar recoger su valor al igual que el valor de todos los demás, no nos devolvía el valor deseado y a veces hacía que dejase de funcionar la aplicación sin saber por qué razón. Después de mirar en la documentación y de buscar información por internet, decidimos implementarlo de diferente forma. Lo que hicimos es no incluir este elemento en el “FormGroup” por lo tanto, no le añadimos el atributo en la vista en el que ponerle el nombre dentro del formulario y no añadimos este elemento dentro de la creación de la variable en el controlador y la variable datos quedaría de la siguiente forma:



Para conseguir el valor del “booleano” del elemento “toggle” lo obtuvimos de una de las formas que ya conocíamos que es utilizando el árbol “DOM” y obteniendo el elemento por su “id” de la vista de la siguiente forma:



Ahora ya habíamos conseguido obtener el valor del elemento, pero, a diferencia de los elementos del formulario anterior, al modificarse, no se actualizaba su valor en el controlador y por lo tanto siempre nos guardaba el mismo valor. Para solucionar este problema, tuvimos que añadirle un evento a este elemento para que cuando el usuario pinchase sobre el elemento y su valor cambiase, llamase a una función que actualizase su valor y así tener siempre actualizado el valor en el controlador para que al guardar los nuevos datos del usuario modificados, siempre guardase el valor actualizado. De esta forma, además, conseguimos que la aplicación ya no dejase de funcionar como lo hacía cuando introducíamos el “toggle” dentro del “FormGroup”. Entonces, el resultado en la vista fue el siguiente:



Como vemos, el elemento abonado tiene un evento llamado “ionChange” que se ejecuta cada vez que este elemento es modificado y ejecuta la función “actualizarAbonado()” que lo que hace es modificar el valor de la variable del controlador que almacena el valor “booleano” que dice si el usuario es abonado o no lo es.

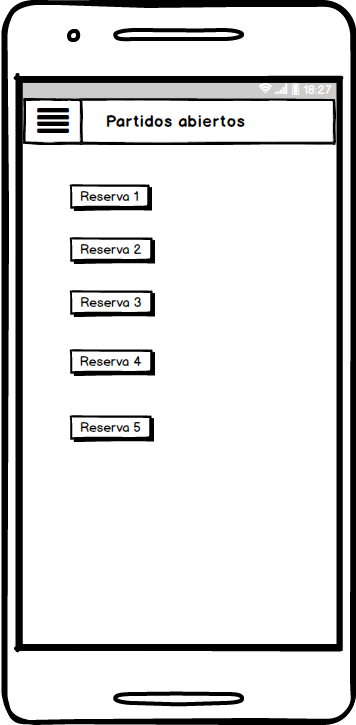
### Exclusiones

Como ya he comentado anteriormente, no hemos conseguido implementar la funcionalidad para el envío de notificaciones entre usuarios. Esto ha sido debido a que es la última funcionalidad que hemos intentado introducir en la aplicación y después de intentarlo durante muchas horas no hemos visto resultados satisfactorios y por lo tanto, hemos hablado con el cliente y hemos acordado no realizar esa funcionalidad porque nos íbamos a pasar mucho de las horas máximas a usar en el proyecto.

Esta funcionalidad, como también he comentado, la íbamos a realizar con el servicio de notificaciones que ofrece *Firebase*. En este caso, sólo la íbamos a utilizar para la plataforma *Android* que es para la que va destinado nuestro proyecto. Para utilizar este servicio, tienes que, una vez compilada tu aplicación de *Ionic* para *Android*, consiguiendo el nombre del paquete que tiene este proyecto *Android*, que lo conseguimos abriendo el proyecto en la aplicación de *Android Studio*. Cuando ya tienes el nombre del paquete añades a tu proyecto de *Firebase* una nueva aplicación de *android* en la que tienes que añadir el nombre del paquete y siguiendo unos pasos que te indica el propio servicio de *Firebase*, consigues añadir ese servicio al proyecto de *Android Studio* y así puedes realizar el envío y recepción de notificaciones entre los usuarios.

Pero, después de seguir todos los pasos que nos indican y de añadir todas las librerías y elementos necesarios para añadir el servicio de notificaciones, nos salían errores en el proyecto que no hemos sido capaces de solucionar.

Después de hablar con el cliente hemos acordado que no se va a implementar el envío de notificaciones, y por lo tanto, ya no tenemos que almacenar en el usuario si quiere o no notificaciones. Además, se ha acordado con el cliente realizar una nueva interfaz para los usuarios en su versione web y móvil, a la cual se podrá acceder desde el menú desplegable que hemos mostrado en los prototipos de las interfaces de la aplicación para usuarios. Esta interfaz va a tener una lista con todas aquellas reservas de menos de cuatro jugadores que se han realizado en la aplicación (ver imagen siguiente). Cuando un usuario se quiera apuntar a un partido pichará sobre una reserva y el sistema le llevará a una pantalla que va a ser la misma pantalla que ya teníamos en los prototipos iniciales que es la de unirse a partido.



# Planificación real

En esta sección se muestra una tabla simplificada con las horas que se habían planificado en la planificación para cada fase del proyecto, al lado aparecen las horas reales invertidas durante la realización del proyecto y por último la desviación de las horas planificadas a las reales.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fase** | **Horas planificadas** | **Horas reales** | **Desviación** |
| 1. **1. Planificación** | **6** | **6** | **0%** |
| **2. Estudio de la tecnología** | **20** | **30** | **50%** |
| **3. Análisis de requisitos** | **29** | **25** | **-14%** |
| **4. Diseño** | **45** | **30** | **-25%** |
| **5. Implementación** | **145** | **180** | **24%** |
| **6. Pruebas** | **10** | **10** | **0%** |
| **7. Redactar memoria** | **30** | **30** | **0%** |
| **8. Reuniones con la tutora de la Universidad** | **15** | **15** | **0%** |
| **Total:** | **300** | **326** | **8%** |

En la planificación hemos utilizado las horas que habíamos marcado para realizarla.

En la fase del estudio de la tecnología hemos pasado las horas planificadas debido a que era una tecnología nueva con la que no habíamos trabajado nunca y nos ha resultado más difícil de aprender de lo que esperábamos a la hora de hacer la planificación.

En la fase de análisis de requisitos hemos utilizado menos horas de las planificadas y esto ha sido debido a que tendríamos que haber puesto menos horas a la hora de planificar.

En la fase de diseño también hemos utilizado menos horas de las planificadas debido a que al utilizar *Firebase* como sistema para almacenar la base de datos y este tener el servicio de base de datos en formato JSON, nos ha llevado menos tiempo el diseño de la base de datos.

En la fase de implementación, la creación de la base de datos nos ha llevado menos tiempo que el que estaba planificado, pero la creación de la funcionalidad del proyecto nos ha llevado mucho más tiempo del esperado debido al desconocimiento inicial de la tecnología y a todos los problemas tenidos con la tecnología como por ejemplo el hecho de las funciones asíncronas. Y, aunque no se haya llegado a implementar y no se refleje ese tiempo en el resultado final, también se ha invertido demasiado tiempo en implementar la funcionalidad de las notificaciones.

En todas las demás fases se han cumplido las horas estimadas en la planificación.

# Conclusiones

Al concluir el proyecto, en cuanto a la funcionalidad que se deseaba inicialmente, no se ha logrado lo esperado, aunque en consenso con el cliente se ha conseguido solventar ese problema y terminar realizando unas aplicaciones que le han gustado al cliente. Por lo tanto, en líneas generales creo que ha sido un trabajo satisfactorio en el que he aprendido nuevos lenguajes de programación y un nuevo framework para la programación multiplataforma que creo que es el futuro de las aplicaciones.

Para el próximo proyecto habrá que estudiar mejor la arquitectura antes de iniciar el proyecto e intentar cuadrar mejor las horas dentro de la planificación para que no haya en algunas fases tantas diferencias de horas como ha habido en el caso de este proyecto.

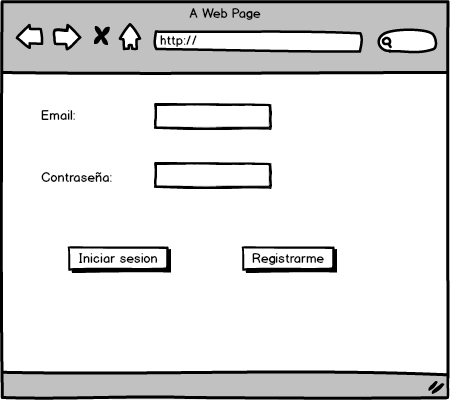
# Anexos

En esta sección aparecen los anexos a la memoria de nuestro proyecto.

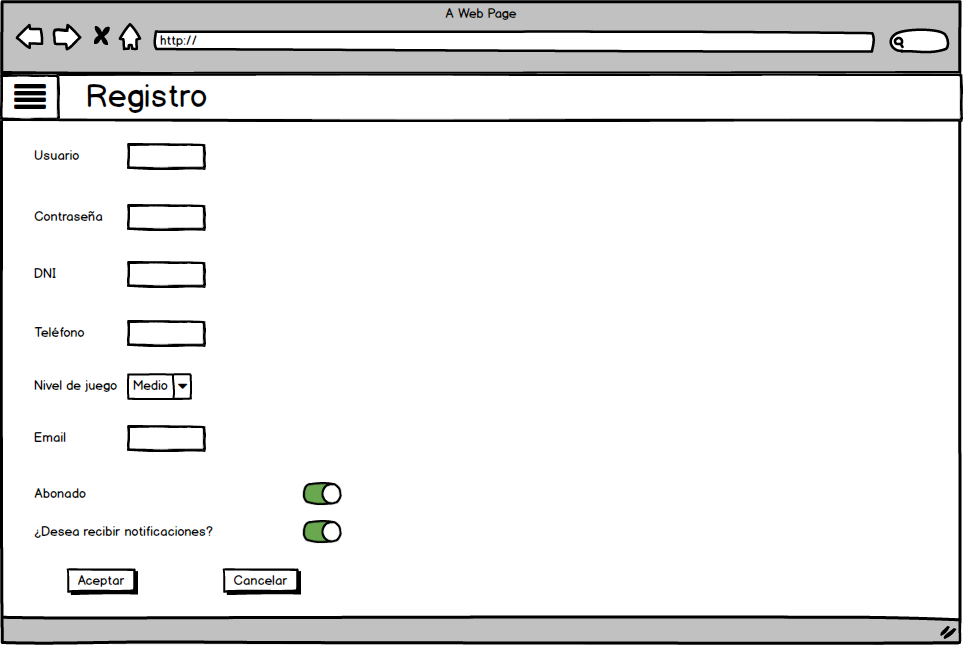
### Prototipos de las interfaces de los usuarios en la versión web

En este apartado se muestran los prototipos de las interfaces de la aplicación para usuarios en su versión web:

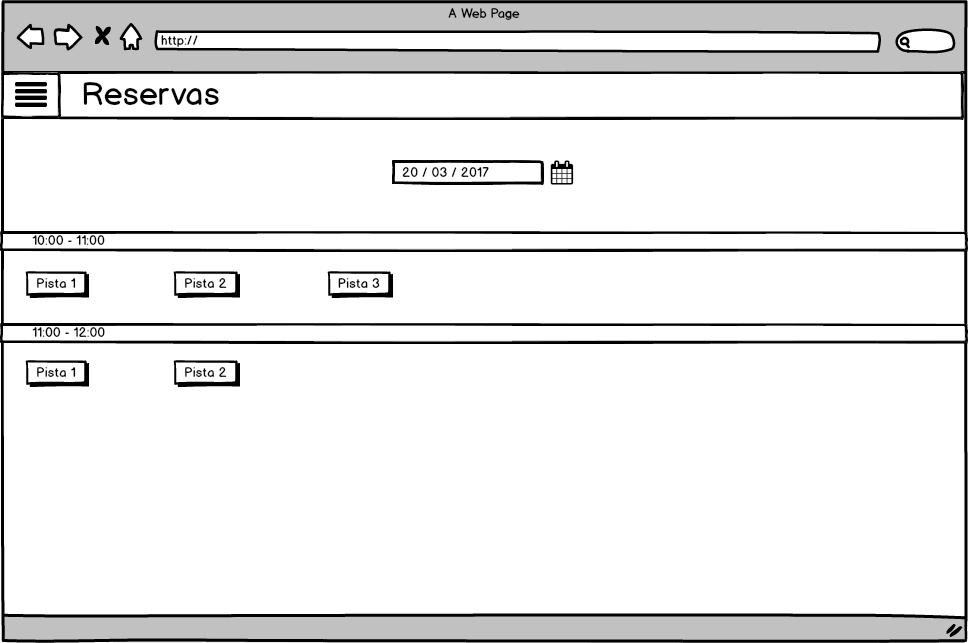
Pantalla de inicio de sesión:



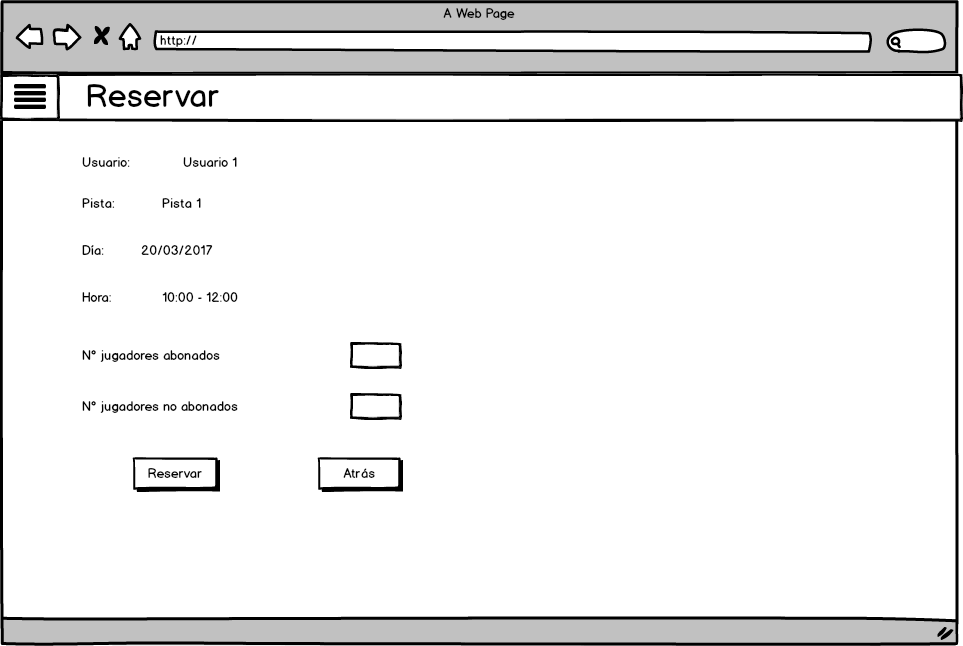
Pantalla de registro:



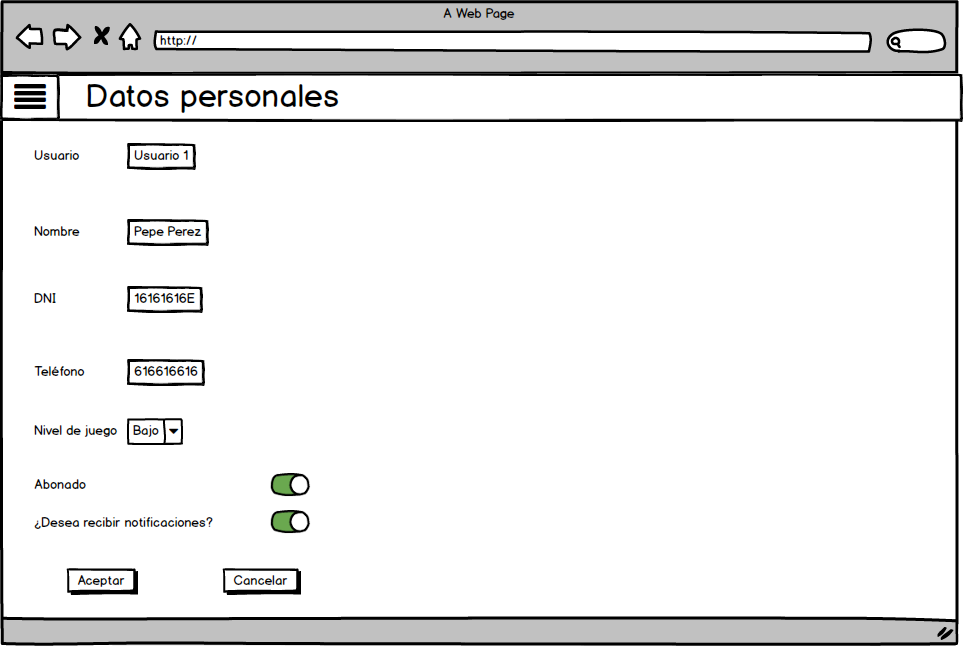
Pantalla para ver las pistas disponibles en los diferentes días y horas:



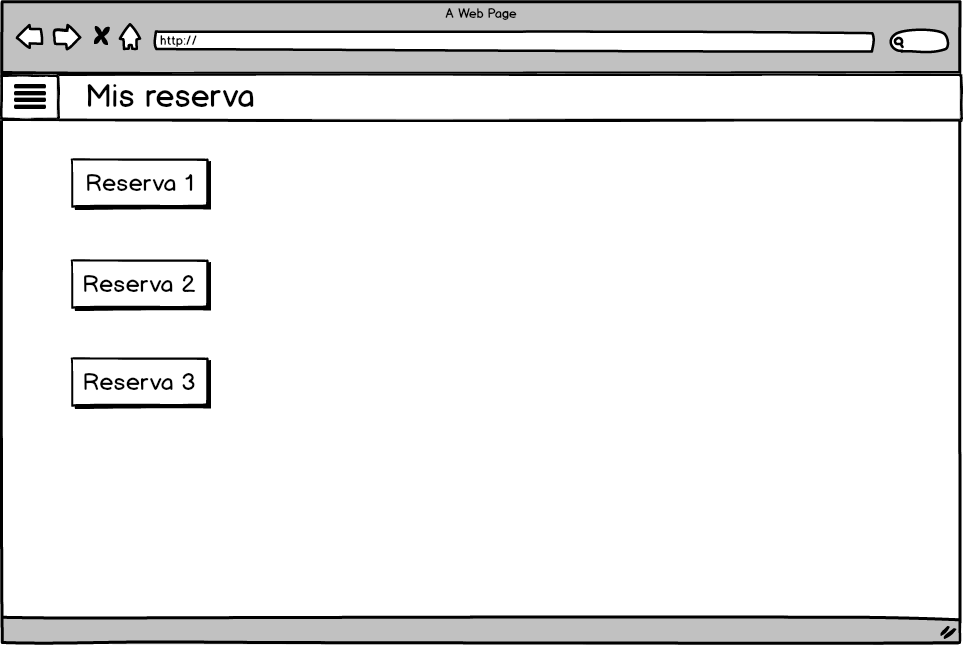
Pantalla para reservar una pista:



Pantalla donde el usuario puede modificar los datos personales:



Y por último la pantalla para ver el historial de las reservas realizadas por el usuario:



# Bibliografía

[1] Página principal de Ionic 2: <https://ionicframework.com/>

[2

] Documentación de Ionic 2: <https://ionicframework.com/docs/>

[3] Página principal de AngularJS: <https://angular.io/>

[4] Documentación de AngularJS: <https://angular.io/docs/ts/latest/>

[5] Página principal de Typescript: <https://www.typescriptlang.org/>

[6] Documentación de Typescript: <https://www.typescriptlang.org/docs/home.html>

[7] Página principal de NodeJS: <https://nodejs.org/es/>

[8] Página de descarga de Android Studio: <https://developer.android.com/studio/index.html?hl=es-419>

[9] Documentación sobre HTML 5: <https://www.w3schools.com/html/html5_intro.asp>

[10] Documentación sobre CSS: <https://www.w3schools.com/css/>

[11] Página principal de Visual Studio Code: <https://code.visualstudio.com/>

[12] Página principal de Github: <https://github.com/>

[13] Repositorio de Github en el que está almacenado el código: <https://github.com/seperev/aplicacion-final>

[14] Página principal de Balsamiq mockups: <https://balsamiq.com/products/mockups/>

[15] Página de descarga de ArgoUML: <https://argouml.uptodown.com/windows>

[16] Página principal de Firebase: <https://firebase.google.com/?hl=es-419>

[17] Documentación de Firebase: <https://firebase.google.com/docs/web/setup?hl=es-419>

1. En este proyecto, y a petición del cliente, partiremos de la premisa de que cada partido de pádel se juega con 4 jugadores [↑](#footnote-ref-1)