Universidad Zaragoza
1542

Gestión de Proyecto Software

Grupo 1: Informe del Primer Sprint (WaitLess)

David Mañas Vidorreta
Ian Paul Ávila Matos
Javier González Galindo
María Lázaro Muñío
Iván Escuín González
Óscar Saboya Gómez
23-11-2018

Índice

1. Introducción

El proyecto se enmarca en la asignatura Gestión de Proyecto Software, obligatoria dentro de la rama de Ingeniería de Software del Grado de Ingeniería Informática de la Universidad de Zaragoza.

Dicho proyecto se basa en la creación de una aplicación web que permitirá reducir el tiempo a la hora de realizar los pedidos en un restaurante. Ésta permitirá realizar el pedido sin esperar a que el camarero llegue gracias a una tablet acoplada a la mesa. La aplicación proporciona una visión clara de la carta, pudiendo ver información de cada plato, incluyendo los alérgenos. Además, facilitará el trabajo a los camareros y cocineros.

1.1 El equipo

El presente proyecto estará desarrollado por el siguiente equipo:

- David Mañas Vidorreta
- Ian Paul Ávila Matos
- Javier González Galindo
- María Lázaro Muñío
- Iván Escuín González
- Óscar Saboya Gómez

Dado que se utilizan los principios de la metodología Scrum, se han tenido que definir diferentes roles:

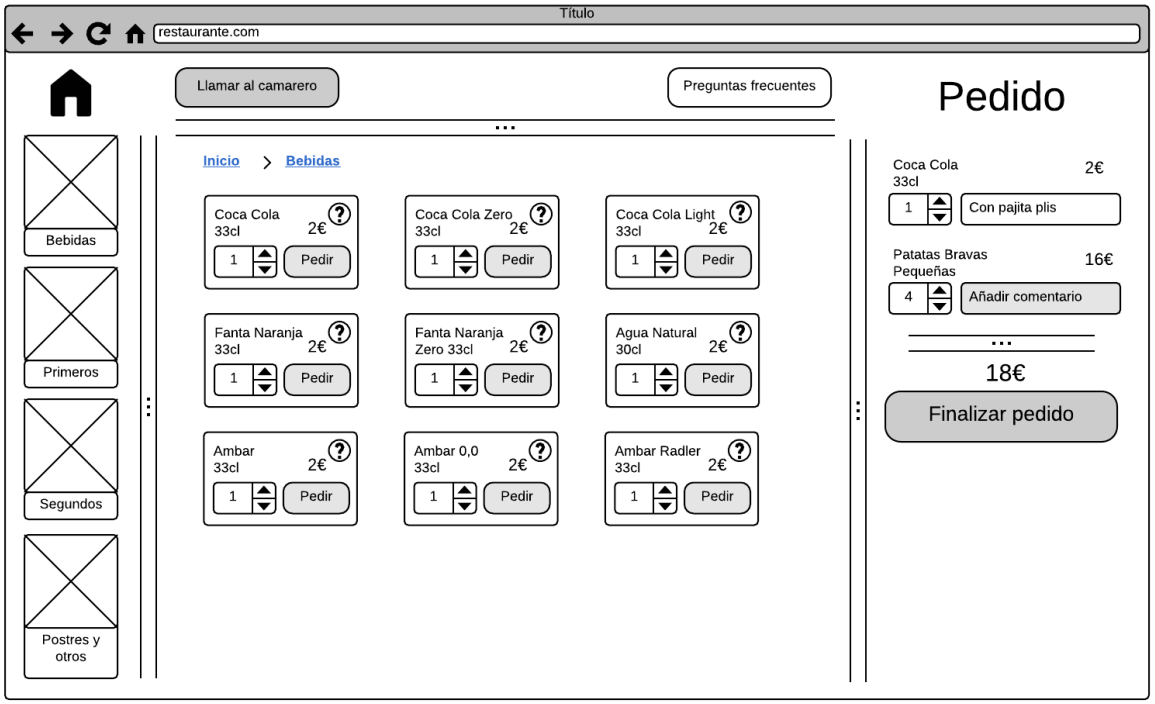
- Scrum Master: Óscar Saboya Gómez
- Dueño de producto: María Lázaro Muñío
- Equipo de Desarrollo: David, Ian Paul, Javier, María, Iván y Óscar

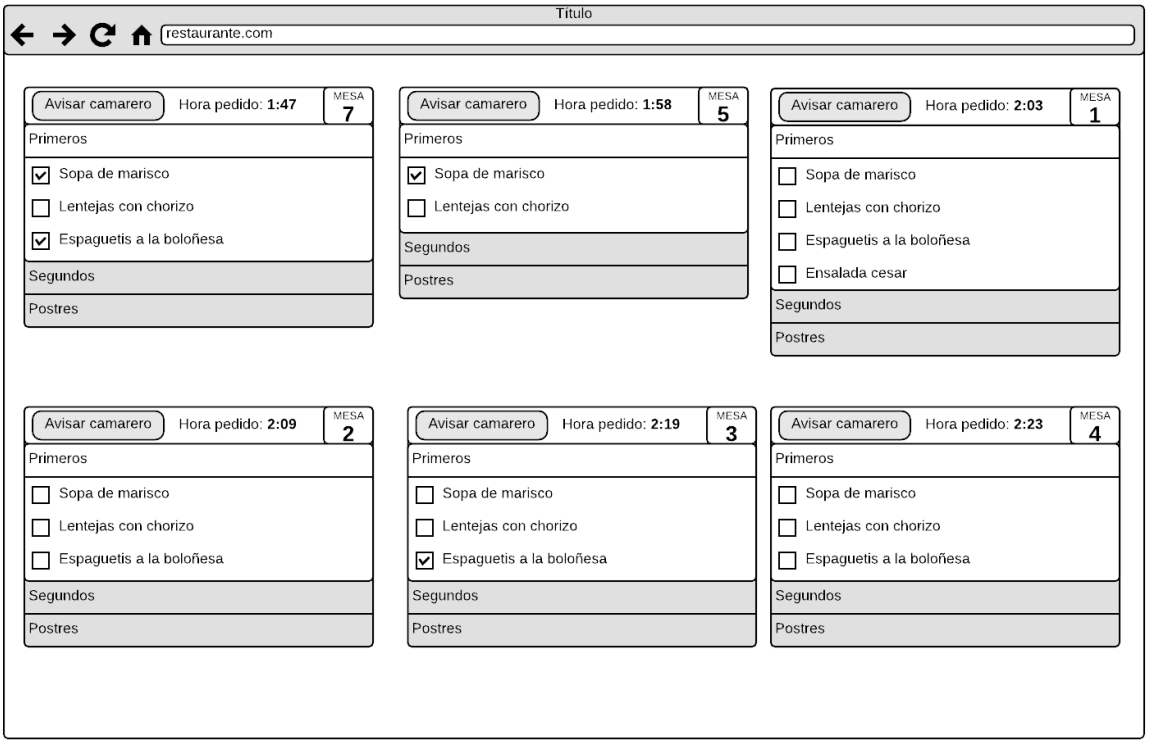
El este documento, se van a tratar todos los temas relacionados con el primer sprint de desarrollo, que se inició el 15 de octubre de 2018 y acabó el 22 de noviembre de 2018. Se dará información sobre el sistema web desarrollado, el proceso seguido para trabajar así como las conclusiones obtenidas.

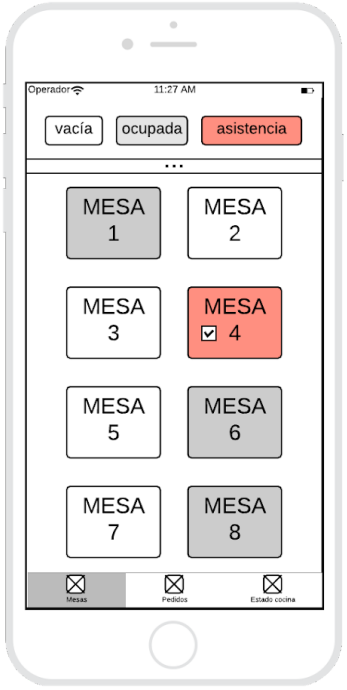
2. Producto

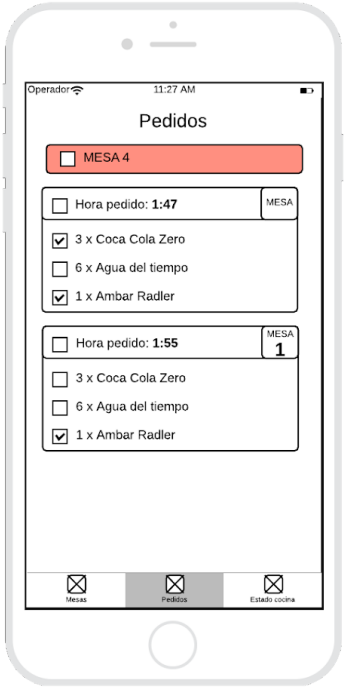
2.1 Plan de producto y planificación

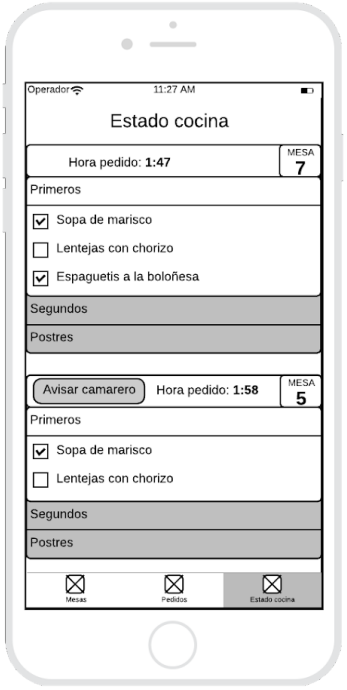
Bocetos de GUI











2.2 Pila de producto

En la tabla mostrada a continuación se observa el estado actual de la pila de producto que se estimaba desarrollar en este sprint. En ella aparecen en verde los PBI que cumplen la definición de hecho, en amarillo los que se encuentran en una etapa avanzada de desarrollo, es decir, solo faltan una o dos tareas para completarlos, y por último en rojo los

PBI que todavía no se ha comenzado a implementar o su estado de desarrollo está poco avanzado.

PBI	Condiciones de satisfacción
Ver Carta (nombre y precio)	Este requisito estará cumplido cuando se pueda ver una lista de todos los productos (platos y bebidas) que componen la carta del restaurante junto con su precio.
Seleccionar producto y cantidad	Este requisito estará cumplido cuando dada una lista de productos se pueda seleccionar uno de ellos e introducir la cantidad que se desea pedir.
Revisar y confirmar pedido	Este requisito estará cumplido cuando se pueda finalizar el pedido y se verifique que se ha realizado correctamente.
Ver pedido actual	Este requisito estará cumplido cuando se pueda visualizar una lista de productos seleccionados junto con sus cantidades y precio.
Modificar pedido	Este requisito estará cumplido cuando, dado un pedido realizado, se puedan eliminar productos, añadir productos o modificar cantidad de mismos y se verifique que realmente los cambios han sido efectuados y son visualizados.
Ver información detallada de cada entrada de la carta	Este requisito estará cumplido cuando, dada una lista de productos, se pueda visualizar los ingredientes de cada uno de ellos.
Añadir comentarios a los productos pedidos	Este requisito estará cumplido cuando se pueda añadir comentarios en cada uno de los productos seleccionados en el pedido.
Consultar pedidos pendientes	Este requisito estará cumplido cuando se pueda ver una lista de pedidos que quedan por realidad junto con su información.
Marcar comida/bebida como servida, preparada, pendiente …	Este requisito estará cumplido cuando en cada pedido se pueda marcar cada producto como pendiente, preparado o servido
Modificar menú	Este requisito estará cumplido cuando se pueda añadir o eliminar bebidas y crear, modificar o eliminar un plato del menú.
Recibir aviso del cliente	Este requisito estará cumplido cuando el camarero pueda ver un aviso que le permita saber qué cliente necesita de su servicio.

La gestión de la pila de producto se ha realizado mediante la plataforma Trello. En ella, se han creado diferentes columnas. Éstas son:

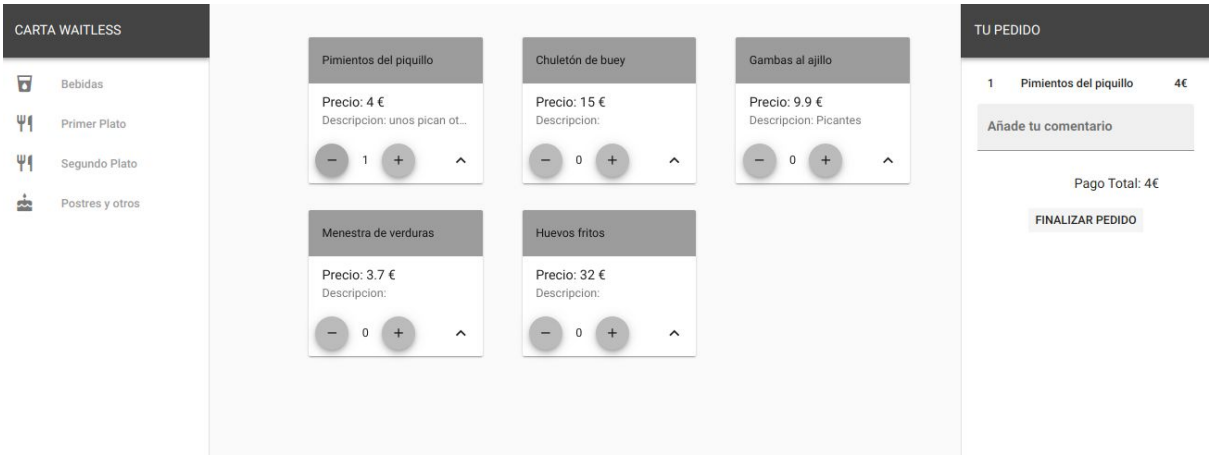
- Pila de producto: Recoge todas las entradas pendientes de hacer y que no se realizarán en el primer sprint.
- Por hacer: Recoge aquellas entradas pertenecientes al sprint actual y que aún no se han empezado a desarrollar.
- En proceso: Recoge las entradas que se están desarrollando actualmente.

- En pruebas: Recoge las entradas que ya están desarrolladas y que falta asociarlas con otra pila del producto o falta realizar los tests correspondientes.
- Hecho (Tarea): Recoge aquellas entradas que ya han pasado los tests y faltan ser aprobados por el dueño de producto.
- Hecho Hecho: Son aquellas tareas que cumplen la definición de hecho consensuada entre todos los miembros del equipo. Ésta incluye los siguientes puntos:
    - Los tests de integración se ejecutan correctamente.
    - Los tests de GUI se ejecutan correctamente.
    - Todas las funciones están comentadas en formato JSDoc.
    - La funcionalidad está desplegada en producción.
    - El trabajo de todos los miembros del equipo de desarrollo está totalmente integrado en cada sprint.
    - El trabajo de cada miembro del equipo ha sido revisado por al menos otro miembro del equipo.
    - Todo el equipo considera que para cada requisito se cumple sus criterios de aceptación.
    - Validación manual del caso de uso.
    - El jefe de producto ha validado y aceptado el requisito.

2.5 Interfaz de usuario

Carta de cliente

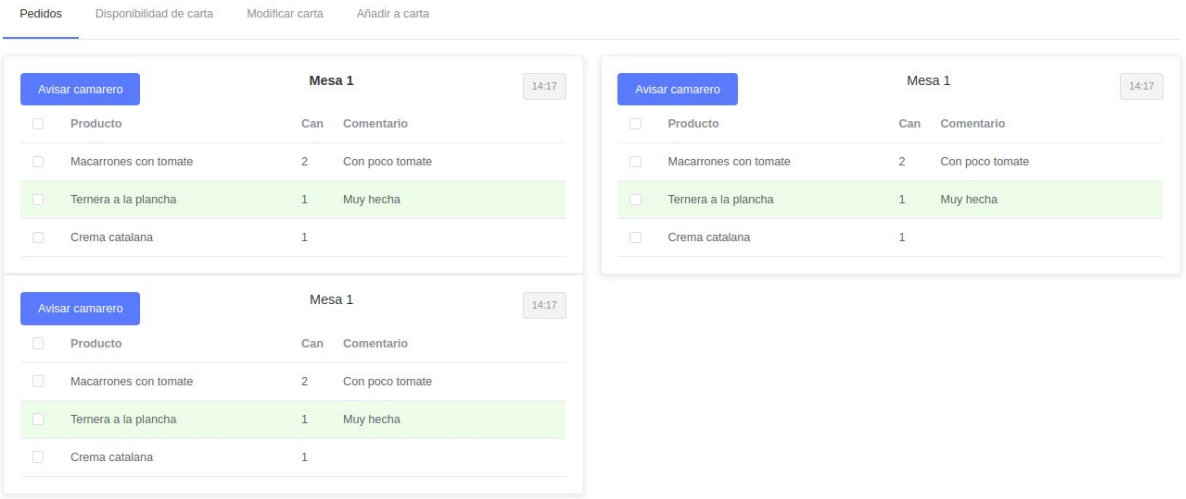
El estado actual de la interfaz de cliente se asemeja bastante al boceto de GUI realizado anteriormente, pese a que es necesario mejorar su estética y usabilidad, la funcionalidad de esta interfaz se encuentra prácticamente completa, a excepción de permitir a los clientes llamar a un camarero. Como aspectos a mejorar en esta interfaz, trabajaremos en el feedback que recibe el usuario al realizar un pedido, ya que es un punto crítico en el uso de nuestro sistema.



Cocina

Pese a que en el boceto inicial se pensó en la GUI de cocina como una única ventana, tras comenzar el desarrollo de las tareas se ha observado la necesidad de ampliar las funcionalidades que se ofrecen mediante el uso de pestañas.

La pestaña por defecto es la de pedidos, donde aparecen las comandas realizadas por cada una de las mesas, con el nombre de los platos, su cantidad y los comentarios que los clientes han aportado. En ella se permitirá marcar los platos como completados así como llamar a un camarero para darle a conocer que uno o varios platos ya están listos para servir. En el momento actual todavía no permite realizar ninguna de estas acciones.

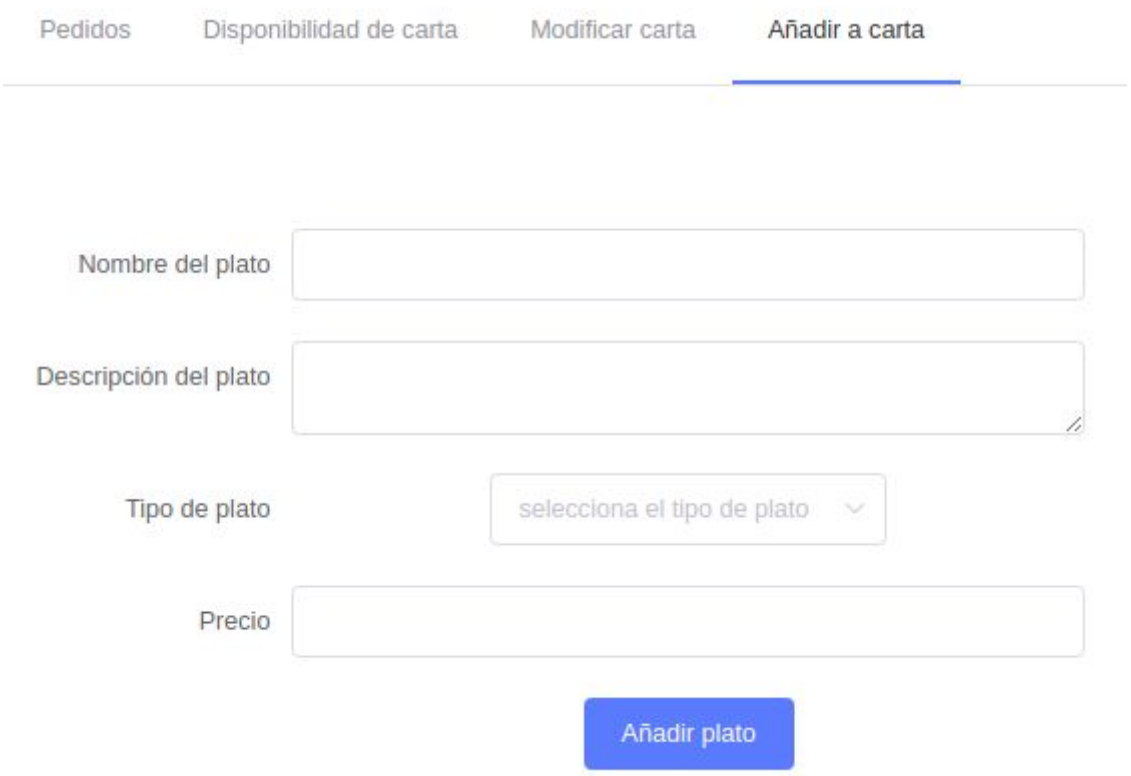


La siguiente pestaña está dedicada a comprobar y modificar la disponibilidad de platos en la carta. Se ha optado por un diseño de tipo lista con un buscador de platos en tiempo real, de esta forma es mucho más rápido localizar el plato o bebida que se desea modificar. En la parte derecha existirá un botón para cambiar la disponibilidad, funcionalidad que todavía se encuentra en desarrollo.

A continuación se ha creado una pestaña que será utilizada para modificar o eliminar un plato o bebida de la carta. La ventana se encuentra dividida en distintas tarjetas que muestran toda la información disponible de un plato, en este momento solo muestra los

productos pero no es capaz de realizar ninguna interacción adicional, en el siguiente sprint el botón *Modificar* nos mostrará un formulario para editar un plato o eliminarlo definitivamente.

Por último se encuentra la pestaña *Añadir a carta*, en la cual se permite añadir un plato nuevo a la carta. Su funcionalidad básica ya se encuentra disponible, pero en la siguiente iteración se añadirá la posibilidad de incluir información adicional como los ingredientes o alérgenos de dicho plato.



Como se puede observar en las capturas de la interfaz mostradas previamente, el diseño actual es simple y está basado en priorizar la funcionalidad en el desarrollo del proyecto, de esta forma será posible obtener valor de este producto antes que si se dedicara más tiempo a perfeccionar el diseño.

Pese a centrarnos en la funcionalidad en la siguiente iteración se prevé mejorar la experiencia de usuario y usabilidad, así como mejorar el feedback recibido por el usuario tras realizar cualquier acción. Si nuestro producto tiene una baja usabilidad es probable que su adopción sea mucho más lenta.

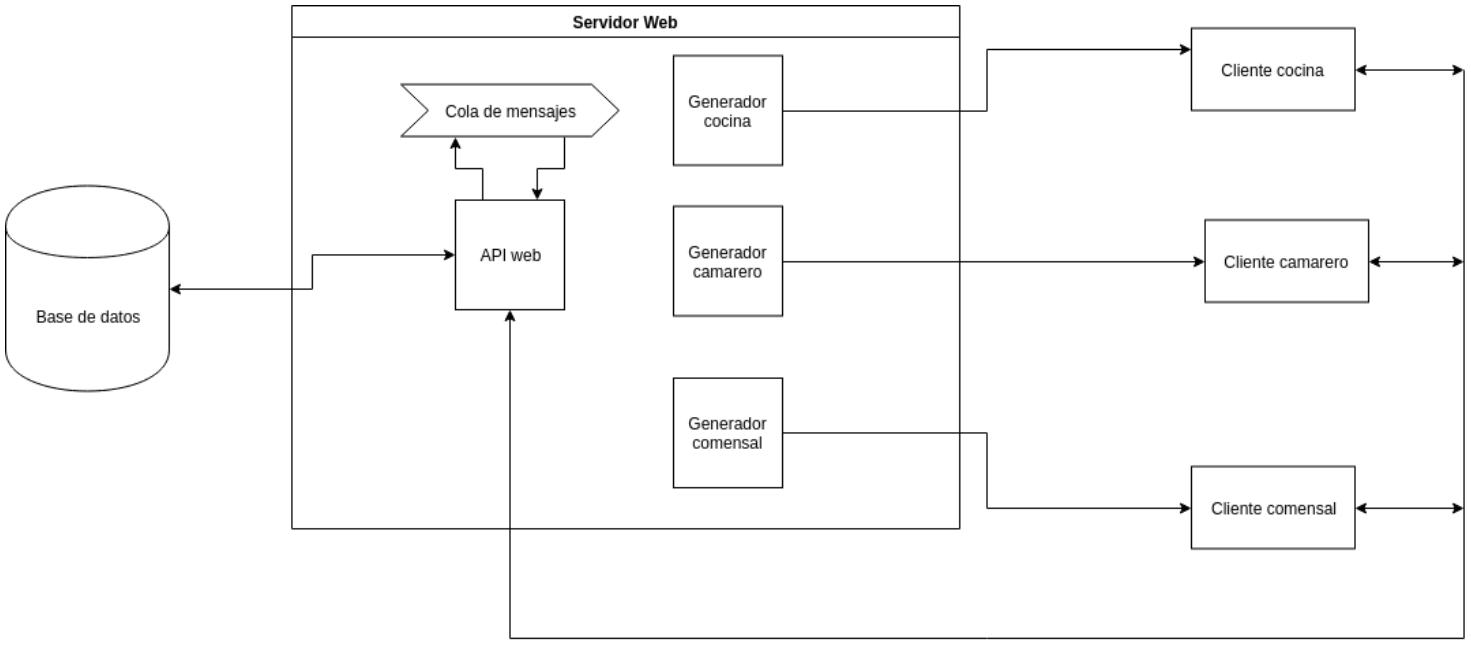
Camareros

La interfaz gráfica que utilizarán los camareros todavía no se ha empezado a desarrollar, en el siguiente sprint se prevé tener una interfaz básica que permita gestionar la bebidas pedidas y las llamadas por parte de las mesas y cocina, con un diseño similar al que se ha mostrado en los bocetos de GUI.

2.6 Arquitectura y diseño

Para el sistema vamos a emplear una arquitectura cliente-servidor de 3 capas, donde separaremos la base de datos, el servidor web y la API que ofrece y el navegador web del cliente. Por ahora la comunicación se establece a través de http entre servidor web y cliente y con https entre servidor web y base de datos, aunque se tiene planeado usar https para todas las comunicaciones. También se ha planteado para alguna funcionalidad del sistema la posibilidad de abrir una comunicación WebSockets entre el cliente y la API web, que estaría alimentada por una cola de mensajes en el lado del servidor (esto se plantea, por ejemplo, para el envío de pedidos en tiempo real desde una mesa del restaurante a la cocina).

Dada la implementación que se ha seguido a lo largo del sprint para la parte cliente e interfaz web del sistema (discutida en el siguiente punto) y la naturaleza del propio sistema, esta es, diferentes vistas y funcionalidades para cada uno de los usuarios, nos hemos visto obligados a separar en varios puertos del servidor web diferentes aspectos: API, cliente de cocina, cliente camarero y cliente comensal.



Esquema general de arquitectura.

2.7 Implementación

Para la implementación de la base de datos se ha usado MySQL, para el servidor web se ha usado Node.js con apoyo de Express para la API web y el framework Vue.js para los clientes.
El choque producido entre la implementación de la API web y los clientes, ambos implementados con nodejs pero de distinta forma, propició los cambios en la arquitectura. Se intentó hacer compatible tanto la implementación de Node y Express con el desarrollo de Vue para tener un solo punto de acceso al servidor Web, pero dado que no se logró, se optó por la solución explicada en el apartado de Arquitectura.

Para facilitar la comunicación de las funcionalidades que ofrece la API web, se ha hecho uso de Swagger para describir los *endpoints* disponibles, y qué parámetros y respuestas ofrecen.

Para el *frontend* se ha optado por usar Vue.js, aunque debido al nulo conocimiento de este framework tras intentar utilizarlo durante unas semanas decidimos desecharlo y comenzar a utilizar AngularJS en su lugar. Tras nuevas dificultades y viendo la mayor complejidad de AngularJS se decidió dedicar tiempo a conocer Vue.js en profundidad y tras ello aplicarlo a nuestro proyecto, tras conocer bien la estructura de dicho framework el diseño de las distintas vistas ha sido relativamente sencillo y rápido; en nuestro proyecto se encuentran totalmente separadas la vista de carta de clientes, la de cocina y la de camareros.

2.8 Despliegue

Por ahora solamente se encuentra desplegada la base de datos usada en desarrollo en Heroku, pero cuando se completen más partes del sistema se desplegará en su totalidad en Heroku para pruebas en desarrollo y en AWS para producción.

3. Proceso

3.1 Testing

En la planificación de este primer sprint, se decidieron hacer tests unitarios, de integración, de GUI y manuales. La realización de estos test se corresponde con su propia tarea para cada entrada de la pila, a excepción de los tests unitarios que forman parte de las tareas de implementación. Además, como no cualquier tipo de test tiene sentido en según qué entradas de la pila, de esta forma se indica cuales son los tests pertinentes a esa entrada.

Para la realización tanto de los test de integración como unitarios, se usará el framework de testing Mocha, junto con la librería de aserciones Chai y el plugin chai-http que permite hacer tests a los diferentes endpoints de la API web en caso de los tests de integración. Para obtener la cobertura de dichos tests se hace uso de la librería Istanbul.

Los tests realizados tendrían esta pinta:

```
//PLATOS

describe('Platos', function() {

    it('Deberia listar todos los ingredientes /ingredientes/<id> GET', function(done) {
        chai.request(server)
            .get('/api/plato/ingredientes/91')
            .end(function(err, res){
                res.should.have.status(200);
                done();
            });
    });


    it('NO Deberia listar ningún ingrediente /ingredientes/<id> GET', function(done) {
        chai.request(server)
            .get('/api/plato/ingredientes/90')
            .end(function(err, res){
                res.should.have.status(204);
                done();
            });
    });


    it('Deberia listar todos los alergenos /alergenos/<id> GET', function(done) {
        chai.request(server)
            .get('/api/plato/alergenos/91')
            .end(function(err, res){
                res.should.have.status(200);
                done();
            });
    });
```

Y al lanzarlos con cobertura, veríamos esta tabla donde se nos dice la cobertura en cada fichero, junto con la cobertura total.

```
  13 passing (1s)

-------------------------|----------|----------|----------|----------|-------------------|
File                     |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
-------------------------|----------|----------|----------|----------|-------------------|
All files                |    73.88 |    45.83 |    63.46 |    76.36 |                   |
 server                  |      100 |      100 |      100 |      100 |                   |
  app.js                 |      100 |      100 |      100 |      100 |                   |
 server/database         |    60.63 |     43.1 |    66.67 |    65.25 |                   |
  querys.js              |    60.63 |     43.1 |    66.67 |    65.25 |... 06,207,208,210 |
 server/modules/carta    |     87.5 |       75 |       80 |     87.5 |                   |
  bebida.js              |    54.55 |      100 |        0 |    54.55 |     6,11,15,18,19 |
  carta.js               |      100 |       75 |      100 |      100 |             10,11 |
  plato.js               |      100 |      100 |      100 |      100 |                   |
 server/modules/pedido   |    61.76 |    33.33 |    28.57 |    60.61 |                   |
  pedido.js              |    61.76 |    33.33 |    28.57 |    60.61 |... 48,54,60,63,69 |
 server/modules/servicio |    71.43 |      100 |        0 |    71.43 |                   |
  avisos.js              |    71.43 |      100 |        0 |    71.43 |               6,9 |
 server/routes           |      100 |      100 |      100 |      100 |                   |
  bebida.js              |      100 |      100 |      100 |      100 |                   |
  carta.js               |      100 |      100 |      100 |      100 |                   |
  pedido.js              |      100 |      100 |      100 |      100 |                   |
  plato.js               |      100 |      100 |      100 |      100 |                   |
  servicio.js            |      100 |      100 |      100 |      100 |                   |
-------------------------|----------|----------|----------|----------|-------------------|
```

En cuanto a los tests de GUI, se planeó usar la herramienta Selenium para su automatización, pero en este sprint no ha realizado ningún tipo de test automático, aunque sí que se han ido realizando test manuales a lo largo del desarrollo de las tareas de Interfaz. Se propondrá pues para el siguiente sprint indagar más en este tema, pues es posible que se necesite alguna herramienta adicional.
La ejecución correcta de los test de tipo pertinente a cada entrada de la pila forma parte de la definición de hecho.

3.2 Construcción automática y control de versiones.

Para facilitar la construcción y puesta en marcha del sistema se hace uso de la herramienta NPM, la cual usamos tanto para la gestión de paquetes y dependencias, como para iniciar cada una de las partes del sistema. Para agilizar el desarrollo, en la API web también se hace uso de la herramienta nodemon, que junto con NPM permite el lanzamiento

automático de la aplicación cada vez que se realizan cambios en el código. Este comportamiento también sucede en el lanzamiento de los distintos clientes, pero a través de las herramientas que trae consigo Vue.js.
Puesto que la gestión de paquetes e inicialización del sistema esta partida en función de cada usuario, se hace uso de la herramienta concurrently para el lanzamiento de los varios scripts en el sistema, manteniendo la posibilidad de ejecutar cada script por separado si se requiere de una monitorización más detallada.

Con respecto al control de versiones en git, se cuenta con dos repositorios principales uno para toda la documentación relacionada con el proyecto, y otro para el desarrollo del sistema. En este último se sigue un workflow concreto mediante el uso de dos ramas principales. En una de ellas, la rama *test* se irán subiendo todos los commits de los distintos miembros del grupo y se resolverán en ella los conflictos que puedan ir surgiendo. Una vez terminado el sprint y cumplida la definición de hecho para cada entrada de la pila terminada se realizará un merge request contra la rama *main*, que recogerá el sistema listo para producción. En la realización de este merge se realizarán los cambios necesarios para deshabilitar las entradas de pila que aún no han sido completadas y que puedan estar presentes en la rama de *test*, por ejemplo comentando código o deshabilitando ciertos scripts en el arranque del sistema. Como por ahora en el primer sprint no se ha tenido en cuenta el despliegue del sistema, excepto en la parte relacionada con la base de datos, es posible que sea necesario modificar el workflow actual a la hora de hacer el despliegue tras cada sprint.

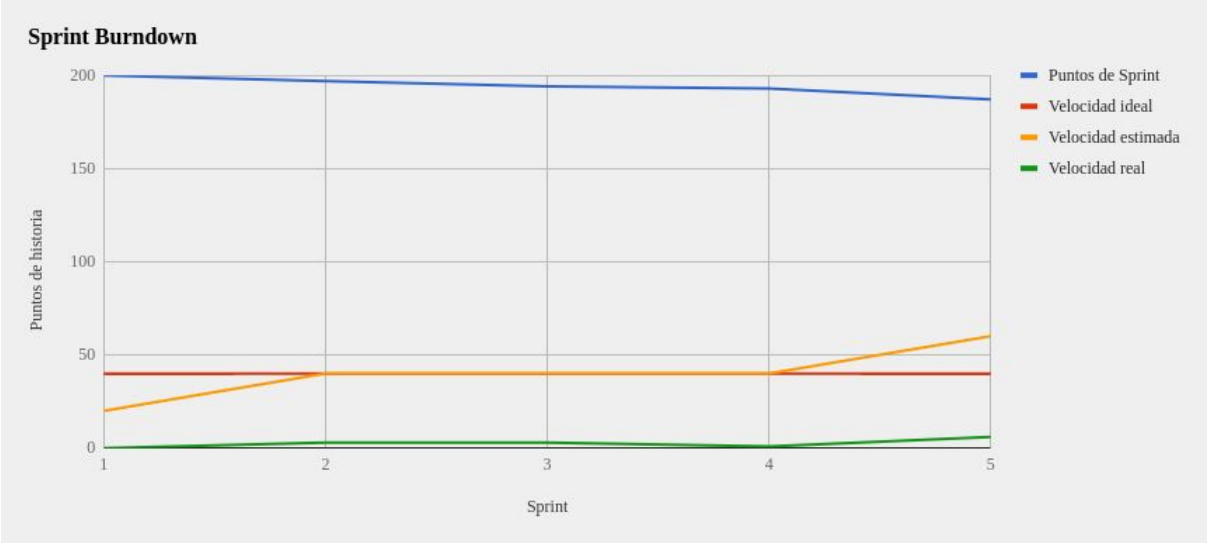
3.3 Definición de hecho

La definición de hecho establecida para el proyecto es la siguiente, pudiéndose endurecer en la siguiente iteración:

- Los test unitarios se ejecutan correctamente
- Los test de integración se ejecutan correctamente
- Los test de GUI se ejecutan correctamente
- Todas las funciones están comentadas en formato JSDoc
- La funcionalidad está desplegada en producción
- El trabajo de todos los miembros del equipo de desarrollo está totalmente integrado en cada sprint
- El trabajo de cada miembro del equipo ha sido revisado al menos por otro miembro del equipo
- Todo el equipo considera que para cada requisito se cumplen los criterios de aceptación
- Validación manual de cada caso de uso
- El dueño de producto ha validado y aceptado el requisito

3.4 Diagrama de burndown y velocidad del equipo

Semanas del sprint	1	2	3	4	5
Fecha inicio	22/10	29/10	5/11	12/11	19/11
Puntos de Sprint	200	197	194	193	187
Velocidad ideal	40	40	40	40	40
Velocidad estimada	20	40	40	40	40
Velocidad real	0	3	3	1	6



Como puede observarse en la tabla superior y en el diagrama de burndown, las estimaciones realizadas no tienen nada que ver con la velocidad real que se ha conseguido, puede deberse a una mala planificación de las tareas a realizar, a una mala estimación de las mismas, a una mala organización o a la suma de todas estas razones.

Se estimaron 200 puntos de historia, y según la definición de hecho establecida apenas se han podido realizar 13 de ellos. Muchas de las tareas a realizar se encuentran cerca de la finalización, pero no sirve como excusa para la entrega de tan poco valor al finalizar el primer sprint.

4. Conclusiones

4.1 Resumen

Claramente, no se han logrado todos los objetivos propuestos para este sprint, aunque sí que hemos atravesado el periodo de adaptación de aplicar Scrum a un proyecto por primera vez, pues el principio ha sido un tanto caótico (Aunque esto más bien que un problema de Scrum, es algo general en los proyectos de cualquier asignatura). A lo largo del desarrollo han ido surgiendo problemas no contemplados en el lanzamiento del primer sprint, y por tanto muchas de las tareas planificadas se han visto afectadas por ello.
También, al ser un grupo relativamente grande, nos hemos centrado más en la realización de tareas en paralelo, en lugar de centrarse en ir acabando entradas de la pila ya empezadas.
Ahora, una vez asentadas las bases del proyecto, y con las ideas un poco más claras, esperamos afrontar el segundo sprint con mayor esperanza de alcanzar los resultados propuestos. Para ello será necesario dedicarle un tiempo adicional a la pila de producto antes de empezar el sprint, pues muchas de las entradas se han quedado en proceso y se debe recalcular su peso en el proyecto en función de su estado de progreso.

4.2 Cumplimiento de objetivos

- Mínimo:
  - **Código en GitHub** -> Repositorio
  - **El producto funciona correctamente e implementa las suficientes funcionalidades** -> Funciona correctamente, pero aún quedan funcionalidades por implementar, incluyendo bastantes del primer sprint.
  - **Informe suficiente y adecuado** -> A la espera de los resultados de este informe
  - **1 de las historias de usuario / requisitos implementados tiene buenos criterios de aceptación** -> Todos los requisitos definidos en el lanzamiento tienen criterios de aceptación
  - **Cobertura de tests unitarios automáticos de 25%** -> Por ahora se cubre el 76.36% **del backend.**
  - **Definición de hecho clara, incluye tests automáticos** -> Definición de hecho
  - **Se lleva un control de esfuerzos** -> Control de esfuerzos en Git. (esfuerzos_g01.ods)
- Notable:
  - **Gestión de dependencias y compilación basada en scripts**-> Se hace uso de NPM tanto para la gestión de dependencias como para la compilación y ejecución del sistema.
  - **50% de las historias de usuario / requisitos implementados tiene buenos criterios de aceptación** -> Todos los requisitos definidos en el lanzamiento tienen criterios de aceptación.