QRest

Introducción

Problemas del sector

- Gran sector de la restauración en españa
- Sector poco automatizado
- Mismos problemas:
  - Toma de pedido:
    - Toman pedido en libreta
    - Costosos sistemas PDA con hardware especializado
  - Carta:
    - No digitalización de la carta
    - Carta guardada como PDF que hay que descargarse
    - Dificil actualizar la carta.
    - No hay flexibilidad de modificar la carta fácilmente.
    - No hay recomendaciones
    - No hay sugerencias
    - No hay facilidad para volver a pedir lo que ya pediste
  - Tiempo de espera:
    - El cliente tiene que esperar a que el camarero pase por la mesa
    - El cliente tiene que esperar para pedir la cuenta
  - No hay registro:
    - No se lleva un registro de los platos más pedidos
    - No se lleva registro informatizado de los pedidos
    - No se hacen estadísticas
  - Web:
    - La mayoría de restaurantes no tienen web
    - Los restaurantes que tiene web tienen que buscar su propio dominio, hosting y mantenimiento.
    - Las web son costosas.
  - Pagos:
    - No suelen tener métodos de pago online.
    - No se suele llevar registro de qué ha pedido cada cliente, va todo la misma cuenta.
    - No se puede a veces pagar por separado.
    - No se puede consultar qué es lo que ha pedido cada persona.

Problemas de soluciones existentes

Las opciones presentes en el mercado suelen caer en las siquientes categorías, pondremos un ejemplo de cada uno:

- Hardware especializado:
  - Pantalla de pedido centralizada:
    - Ejemplo: McDonals

        * Problemas:
            · Súper costoso y especializado.
            · Solo lo tienen actualmente franquicias.
            · Solo funciona para modelo de recoger en barra.
    - Hardware móvil:
        * Ejemplo: PilarBox
        * Problemas:
            · Hardware costoso
            · Hay que cargar los dispositivos.
- Web + QR:
    - QR estático:
        * Variantes: En la mesa, tarjetita o en el servilletero.
        * Ejemplo: Qamarero. (En tarjetitas asociadas a una mesa)
        * Problemas:
            · Seguridad: Cuando vuelves a casa si sigues teniendo el QR o el enlace puedes pedir desde casa a la mesa aunque ya no estés allí.
            · Si están en la mesa y las mesas son modulares hay que ponerse en consenso de qué QR usar pedir.
            · Si las mesas se mueven ocasiona confusión para los camareros de donde está la mesa.
    - QR por pedido:
        * Funcionamiento: Cada vez que entra un cliente al restaurante se genera un QR para ese pedido.
        * Ejemplo: Yasaka, Servicio QbaR
        * Problemas:
            · Tener que estar generando un QR nuevo por cada cliente (grupo de clientes) que entra.
            · Tener que atender a cada cliente que entra.
    - QR estático + contraseña:
        * Ejemplo: Sushi Som
        * Funcinamiento: Genera una contraseña diaria que el cliente tiene que usar.
        * Problemas:
            · Los mismos que los QR estático con la diferencia que el problema dura lo que dure la contraseña.
            · Incomodidad para el cliente, el tener que estar preguntando la contraseña.
            · Los camareros tienen que estar informando de la contraseña.

Como se puede observar no hay una solución lo suficientemente satisfactia que solucione todos los problemas. Todos tienen algunas ventajas e inconvenientes. Es por todos estos problemas expuestos que muchos restaurantes son reacios a este tipo de sistemas, o no pueden costearselo.

Lista de opciones:

- Servicio QbaR: servicioqbar
- PilarBox: pilarbox.com
- McDonalds: mcdonalds.es
- Yasaka: yasaka.es
- Qamarero: qamarero.com
- SushiSom: sushisom.net

Motivación

Aquí plasmo por una parte la motivación personal y la motivación técnica al respecto.

Por la parte personal: Todo comenzó un día yendo a un sushi llamado Yasaka, que tenía un sistema web + QR del tipo QR por pedido, pero que era un desarrollo específico para ese restaurante pero con muchas carencias, entre ellas: frontend poco atractivo, no se sincronizaban los pedidos y no se podía pagar por separado. Desde entonces mi mente empezó a darle vueltas y a decirme a mí mismo: Oye esto se puede hacer para todos los restaurantes. Desde entonces me puse a ver lo que había ya hecho en enero de 2023. Por ese entonces no había muchas soluciones o no las encontré. Y desde que decidí realizar el proyecto he ido descubriendo todas las opciones que hay en el mercado y que se han expuesto en el apartado anterior. Muchas de ellas han surgiodo este 2023. Otras motivaciones personales.

- También era una oportunidad de aprender desarrollo web, que es lo que más me interesa, abordando un problema que me intersa mucho.
- Es un trabajo muy interesante ya que es algo que todo el mundo experimenta, en mi caso como cliente de restaurantes.
- Como motivación también tenía el hecho de que soy emprendedor y la idea es sacarlo como negocio tras el TFG.
- Quería dearrollar una aplicación full-stack
- Aprender todo lo aprendido en la carrera de Ingeniería del Software
- Ponerme a prueba a mí mismo para desarrollar una aplicación que combinase tantas coasa que desconocía.
- Aprender cómo hacer ingeniería del software con un proyecto tan grande.

Por la parte técnica:

- Este tipo de soluciones son muy muy recientes. Ma mayoría de soluciones ha surgido este 2023. Por lo que es era un campo de investigación e innovación.
- Hay muchísimas oportunidades de mercado, ya que este tipo de soluciones no están extendidas.
- Es una solución súper escalable a todos los restaurantes.
- Una misma aplicación puede solucionar muchísimos de los problemas que los restaurantes experimentas. Sobre todo los pequeños restaurantes y los nuevos.
- Es un proyecto full stack.

- Combina muchos aspectos técnicos:
  - Transacciones en base de datos
  - Sincronización entre clientes con websocket
  - Bases de datos NoSQL.
  - Modelos de datos capaces de representar toda la casuistica que puede darse en un restaurante real.
  - Sincronización de servidores.
  - Interacciones con JavaScript.
  - Eventos con JavaScript.
  - Guardar datos del usuario en LocalStorage sin que lleguen los datos del cliente al servidor y por tanto con privacidad.
- A nivel de ingeniería:
  - Desarrollar una aplicación en un campo que está todavía emergiendo. Y por tanto hay poco en lo que te puedas basar.
  - Metodologías de desarrollo software.
  - Requisitos, Casos de uso, Casos de prueba, tests.
  - APIs, POO en Python y JS.
  - Frameworks de desarrollo web
  - Aprender a estructurar un proyecto web grande.
  - Refactoring.

Objetivos

Vamos a diferenciar en tres tipos de objetivos: objetivos personales, profesionales y del proyecto.

Objetivos personales:

- Hacer un TFG por que que estuviese realmente emocionado de hacer.
- Montar un negocio a partir de mi TFG
- Desarrollarme tanto personal como profesinalmente.
- Aprender desarrollo web.
- Hacer una aplicación interesante full-stack para mi porfolio.

Objetivos profesionales:

- Aprender frontend: JS y CSS
- Aprender sincronización con Websockets.
- Aprender profundamente MongoDB.
- Aprender desarrollo web.
- Aprender transacciones en base de datos.
- Aprender sobre arquitectura software aplicada a web.
- Aprender a hacer APIs.
- Aprender CSS y JS
- Aprender POO en Python y JS.
- Aprender a seguir una metodología ágil en un proyecto grande.

Objetivos del proyecto: Solucionar los problemas encontrados en las soluciones

existentes.

- Sincronización de pedidos de clientes.
- Pagos por separado.
- Carta digital flexible y dinámica.
- Sugerencias de clientes.
- Privacidad de datos del cliente.
- Poder manejar toda la casuistica de los restaurantes.
- Crear una gestión de comandas por parte del personal del restaurante que sea flexible.
- Proponer una solución a los QR estáticos y por contraseña.
- Minimizar el hardware necesario por parte del restaurante.

Solución buscada

Se busca una solución que resuleva los siguientes problemas:

Por parte del restaurante:

- Errores al tomar comandas.
- Tiempo de camareros destinado a tomar comandas.
- Tiempo de camareros destinado a llevar la cuenta.
- Tiempo de camareros destinado a cobrar a los clientes.
- Errores en el flujo de trabajo no automatizado.
- No disponer de web.
- No carta de digitalizada.
- Poca flexibilidad para modifcar la carta.
- No se lleva registro informatizado de los pedidos.
- Pedidos a domicilio por telefono.
- Reserva de mesa por teléfono.
- Dificultad para ver el estado de los pedidos de una mesa.

Por parte del cliente:

- Tiempo de espera para pedir comanda.
- Tiempo de espera para pedir cuenta.
- Tiempo de espera para pagar.
- Llamar para pedir a domicilio.
- Llamar para reservar mesa.
- Malentendidos al pedir.
- Dificultad para dividir la cuenta.
- Dificultad para pagar por separado.
- Dificultad para recordar el total de lo que se ha pedido y lo que no.
- Dificultad para ver el estado del pedido.

Requisitos de la solución buscada:

Se busca una solución integral que pueda abordar todas estos problemas de una forma accesible económicamente, sobre todo sin una barrera de entrada inicial,

de forma flexible y poco intrusiva.

- Digitalización de carta flexible.
- Pedidos por parte del cliente.
- Sistema de gestión de comandas flexible. (De modo que el restaurante no tenga que adaptarse a un flujo concreto, sino que la solución se adapte al flujo de funcionamiento que ya tiene el restaurante).
- Registro de todos los pedidos.
- Página web.
- Recomendaciones por parte del restaurante.
- Recomendaciones al cliente para pedir lo que se ha pedido anteriormente.
- Pagos por separado.
- Pagos online.
- Pagos en caja.
- Reserva de mesa.
- Pedidos online.
- Sincronización de pedidos de clientes de la misma mesa.
- Privacidad de datos del cliente.
- Ver estado del pedido.
- Pedidos flexibles.
- Bajo coste.
- Mínimo hardware.

Puntos fuertes QRest vs competidores

Diseñado pero no implementado

- Solución segura y cómoda de generación de pedidos con QR.
- Flujo flexible de gestión de comandas del restaurante.
- Conexión con la impesora de tickets con librería de JS.
- Búsquedas en la carta, con filtros.
- Frontend con elementos complejos.
- Frontend para modificar la carta.
- Llamar al camarero desde la app.
- Sincronización de estado de servidores.

Producto

- Pedidos por parte de cliente.
- Privacidad de datos de los clientes.
- Recibos y pagos de forma individual o total.
- Recomendaciones de repetir lo pedido anteriormente.
- Sugentencias del restaurante en base a etiquetas.
- Pagos en caja con confirmación de pago.
- Sincronización de pedido de clientes de la misma mesa.
- Flexibilidad para modificar la carta.
- Seguridad de API.
- Mostrar alérgenos.

Ingeniería

- Arquitectura en capas al estilo clean Achitecture.
- Tests de integración.
- Estructura de datos capaz de representar toda la casuística de restaurantes.
- Transacciones seguras de casos de uso.
- Metodología ágil basada en sprints.
- Uso de LocalStorage para guardar los datos del cliente sin que lleguen al servidor.

Tecnología y herramientas

Lenguajes de programación

- Python: Se ha utilizado para desarrollar todo el backend de la aplicación, desde desarrollo de la API, casos de uso, tests, hasta llamadas a la base de datos.
- JavaScript: Se ha utilizado para gestionar el HTML y CSS de la página web. También para hacer que la página sea interactiva con eventos, etc. Se han implementado patrones como Modelo-Vista-Controlador y POO en JS.

Tecnologías de desarrollo web

- HTML: Para generar las páginas de la web.
- CSS: Para dar estilo a las páginas de la web.

Frameworks

- BulmaCSS: Se ha usado como el framework de CSS usado para que la aplicación sea responsive y orientada a mobiles.
- FastAPI: Se ha utilizado para definir la API de de la aplicación.
- Pytest: Se ha utilizado para hacer los test de integración.
- Jinja2: Se ha utilizado para la generación dinámica del HTML, CSS y JS.

Tecnologías de bases de datos

- MongoDB: Se ha utilizado como base de datos No-SQL para persistir todos los datos del dominio de la aplicación.
- LocalStorage: Se ha utilizado para almacenar los datos del cliente desde el JS sin que pasen por el servidor.

Herramientas de desarrollo

- Visual Studio Code: Se ha utilizado principalmente para la documentación tanto para escribir como para visualizarla con sus extensiones. También generación y visualización de prototipos, de toda la parte web, HTML, CSS y JS.

- Pycharm: Se ha utilizado como la herramienta principal para desarrollar el código del backend en python. También para desarrollar y correr los tests. También para correr el servidor y debuguear.

Herramientas de gestión

- Git: Se ha utilizado para llevar el control de versiones de la aplicación y documentar los cambios.
- GitHub: Se ha utilizado como repositorio del proyecto.
- Trello: Se ha utilizado para definir, documentar y cambiar de estado las tareas de los sprints.
- MongoDB Atlas: Se ha utilizado como repositorio en la nube de nuestra base de datos mongoDB. Y para crear documentos, eliminarlos, modificarlos, etc.
- Googel Meet: Para las reuniones online de seguimiento del TFG.
- Cloudinary: Para guardar en la nube las imágenes que el proyecto ha necesitado. Como: imágenes de elementos de la carta, iconos de la aplicación (ej: alérgenos, secciones, etc.).
- Pandoc: Se ha utilizado para generar PDFs a partir de los archivos Markdown.
- Calendario a papel: Para organizar la semana y saber qué tarea hacer en cada momento.
- Google Calendar: Para guardar las reuniones de seguimiento del TFG.
- Miniconda: Para gestionar el entorno de python y sus librerías.
- Heroku: Para desplegar la aplicación.
- Zathura: Para visualizar los PDF.

Herramientas de documentación

- Confluence: Para hacer los reportes de los sprints y otra documentación como investigaciones de competidores, análisis de posibles tecnología a usar.
- Papel y boli: Mucha de la documentación, sobre todo la menos formal ha sido con papel y boli. Cada vez que trabajaba en el proyecto tenía papel y boli al lado e iba haciendo bocetos, lluvia de idea, definir los objetivos de los pomodoro, del día, etc.
- Neovim: Como blog de notas de usar y tirar. También para escirbir los archivos markdown.

Herramientas de diseño

- MermaidJS: Para definir textualmente todo tipo de diagramas de la aplicación. Compatible con Markdown. Y visualizarlo.
- Bocetos y diagramas a papel: La mayor herramienta de diseño ha sido bocetos y diagramas de flujo en papel.

Recursos adicionales

- Websockets: Es la forma de sincronizar el frontend con el backend y de sincronizar las vistas de los usuarios.

- Uvicorn: Es el servidor que se ha usado para correr la aplicación.

Metodología

Resumen de la Metodología

- Objetivo: Definir y desarrollar el proyecto, establecer bases de actuación, y detallar decisiones tomadas.
- Enfoque: Adaptación a cambios, actualizaciones, y variaciones de requisitos.

Metodología Ágil

- Uso de la metodología ágil Scrum para facilitar adaptaciones rápidas a cambios durante el desarrollo.
- Roles: Yo, (el estudiante) ha tomado todos los roles: diseñadores, analistas, desarrolladores, testers; excepto el de Product Owner (cubierto por el tutor).
- Iteraciones:
  - Duración de una o dos semanas.
  - Flexibilidad en la duración, debido a inprevistos en el desarrollo y pausas entre iteraciones debido a causas externas.
  - Iteraciones orientadas en funcionalidades concretas.
- Reuniones: Tras cada iteración para discutir avances y planificar la siguiente.

Planificación y gestión del proyecto

Se han usado las siguientes tecnologías y herramientas: Trello, Confluence, Calendario y papel y boli.

1. La documentación formal y definición de objetivos de sprints con sus funcionalidades se definen en Confluence.
2. Luego para cada sprint se ha creado una lista en Trello con todas las tareas. Al iniciar el sprint se van pasando a To Do y siguen el flujo de To Do, Doing, Done. Y al finalizar el sprint todas las de Done se pasan a la tabla del sprint otra vez.
3. Luego se estiman las horas para cada tarea y las prioridades.
4. Luego en base a la prioridad y el tiempo que consumen se planifica en un calendario a papel semanal.
5. Luego dentro de cada día y con cada tarea concreta se establecen objetivos del día y objetivos para cada pomodoro a papel, que una vez completado se

desecha. Los pensamientos, bocetos, etc. también a papel, por comodidad y rapidez.

Nota: las cosas más importantes se han ido documentando en Confluence, en Trello en la descripción de tareas concretas, o en el propio repositorio.

Esta metodología de trabajo se ha ido consolidando a lo largo del desarrollo de la aplicación.

Al principio esta metodología no se seguía tal y como se ha explicado, sino más bien de una forma un poco más laxa. En los últimos meses del proyecto al seguir esta metodología se ha avanzado mucho más rápido.

La mayor mejora se ha dado al definir el calendario semanal y planificar los objetivos de cada pomodoro a papel.

Sprints

- Al principio se planificaron los sprints de los 3 primeros meses en detalle, definiendo lo que se iba a hacer cada semana.
  - Pero los siguientes factores lastraron muchísimo el desarrollo inicial rompiendo completamente con la planificación:
    - No sabía mucho sobre las tecnologías a utilizar
    - Se replanteaba continuamente las funcionalidades de la aplicación y el enfoque.
    - Surgían errores y necesidades imprevistas.
  - Desde entonces lo que hicimos es definir las distintas funcionalidades del sistema y un orden de implementación, pero en vez de planificarlo a largo plazo, se iban definiendo a dos semanas vista.