Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación
IIC2143 - Ingeniería de Software (II/2019)

Proyecto Semestral

1. Objetivos

- Aplicar metodologías ágiles en el contexto de un equipo de desarrollo.
- Aprender el *framework Ruby on Rails* para desarrollar aplicaciones web.
- Aprender a usar distintas herramientas por cuenta propia y explorar distintas soluciones dependiendo de las exigencias del cliente o *product owner*.
- Conocer sobre buenas prácticas y herramientas de desarrollo de *software*.

2. Introducción

Hoy en día el número de restaurantes/locales de comida que existen es enorme. Desafortunadamente muchos tienen el problema de que no son capaces de promocionarse adecuadamente, por lo que después de un tiempo al no lograr alcanzar una buena cantidad de clientes se ven obligados a cerrar. Por otra parte, la tecnología ha hecho posible que las personas puedan realizar una gran cantidad de tareas, como ir de compras, sin ni siquiera tener que salir de sus casas.

En este contexto, a tu equipo de trabajo le han solicitado realizar una nueva e innovadora aplicación web, donde los clientes puedan obtener información acerca de distintos restaurantes y realizar pedidos que sean entregado en el lugar que deseen.

3. Características Generales

La aplicación debe permitir a los visitantes buscar restaurantes, acceder a su menú y seleccionar distintos platos para comprar.

Una vez que un cliente se registra e inicia sesión, este podrá realizar una compra. La compra se define como una orden o pedido y puede ser de uno o varios platos del restaurante (que se pueden repetir) y declarando una dirección de entrega (el usuario la puede tener predeterminada o cambiarla si lo desea). Es importante mencionar que si bien se está realizando una compra, el pago no se realiza mediante la aplicación, sino que es presencial, aunque sí debe ser capaz de definir el medio de pago (Redcompra, efectivo, etc).

Por otra parte, la aplicación debe permitir la creación de nuevos restaurantes. Para poder crear un restaurante se debe llenar un formulario, solicitando ser parte de la comunidad de restaurantes. Esta debe ser revisada por un administrador de la aplicación, y puede ser aceptado o rechazado. El restaurante podrá editar su menú agregando, editando o eliminando platos.

3.1. Tipos de usuario

Su aplicación debe manejar los siguientes tipos de usuario:

- **Visita:(No registrado en la página)** Puede buscar y ver la información de los distintos restaurantes (menú, comentarios, likes, etc.). No puede realizar pedidos.
- **Cliente (Un usuario registrado en la página):** Puede buscar y ver la información de los distintos restaurantes (menú, comentarios, likes, etc.). Puede realizar pedidos. Puede hacer comentarios acerca del restaurante al que le ha hecho un pedido.
- **Administrador general:** Tiene la capacidad de aceptar/rechazar solicitudes de restaurantes. Además puede eliminar clientes/restaurantes según los criterios que deben ser definidos por el grupo de trabajo.

3.2. Comportamiento General

- La aplicación debe permitir el registro y la autenticación de usuarios.
- Un usuario debe poder buscar restaurantes cercanos.
- Un usuario debe poder ver la información de cada restaurante, así como también el menú y las valoraciones que otros usuarios han realizado.
- Un usuario registrado debe poder modificar los datos de su cuenta e incluso eliminar esta.
- Un usuario registrado (cliente) debe poder generar una orden de uno o más platos a un restaurante específico. Opcionalmente podrá agregar una nota.
- Un usuario registrado (cliente) debe poder cancelar una orden siempre y cuando el restaurante aún no la haya aceptado.
- Un usuario registrado (cliente) debe poder dejar un comentario y evaluar un restaurante donde haya comprado comida antes.
- Un usuario registrado (cliente) debe poder ver el historial de compras que ha realizado en la aplicación.
- Un usuario registrado (cliente) debe poder marcar restaurantes como favoritos.
- Un restaurante debe poder crear, editar y eliminar los platos de su menú.
- Un restaurante debe poder ver las órdenes de compra recibidas.
- Un restaurante debe poder aceptar/cancelar/dar por finalizada una orden enviada por un cliente.
- Un restaurante debe poder ver un resumen de todas las compras realizadas en el local.
- Un usuario registrado (administrador) debe poder moderar los comentarios en los restaurantes.
- Un usuario registrado (administrador) debe poder aceptar o cancelar el registro de un restaurante.

4. Atributos mínimos

4.1. Usuario

Debe manejar al menos los siguientes aspectos de un usuario:

- Nombre

- Correo
- Imagen de perfil
- Dirección
- Restaurantes favoritos
- Registro de pedidos realizados

4.2. Restaurante

Debe manejar al menos los siguientes aspectos de un restaurante:

- Nombre
- Correo
- Dirección
- Valoración
- Imagen(es)
- Numero de teléfono

4.3. Plato

Debe manejar al menos los siguientes aspectos de un plato:

- Nombre
- Precio
- Descripción.
- Cantidad de personas sugerida.
- Imagen(es)
- Valoración (Likes)

4.4. Orden

Debe manejar al menos los siguientes aspectos de una orden:

- Plato(s) pedido(s) en la orden.
- Precio final a pagar.
- Medio de pago.
- Dirección de envío de la orden.
- Hora estimada de entrega.
- Estado del pedido.
- Notas/información adicional de la orden.

4.5. Comentario

Se debe manejar al menos los siguientes aspectos de un comentario:

- Restaurante/Plato al que responde
- Usuario
- Fecha y hora de creación
- Contenido
- Puntaje de reputación

4.6. Inscripción de restaurantes

Las decisiones de modelación de un restaurante, en cuanto a cómo se administran, quedan a libre criterio de cada grupo, sin embargo se deben cumplir con los siguientes requerimientos mínimos:

- Contar con los atributos mínimos mencionados en la sección **4.2**.
- Debe ser administrado por un usuario debidamente registrado.
- Deben ser creados mediante una solicitud que luego es aprobada o rechazada por un administrador de la aplicación.
- El restaurante no puede ser visible sin previa aprobación.
- El administrador del restaurante debe poder editar el menú creando, editando y eliminando platos. Además, debe poder aceptar o rechazar pedidos.

5. Información que se puede proporcionar

Pueden usar su creatividad pero aquí se ofrecen algunas ideas:

1. Usuarios pueden filtrar los restaurantes según popularidad.
2. Usuarios pueden filtrar los platos según popularidad.
3. Administrador puede crear restaurante y asignar un encargado.
4. Pagar mediante la aplicación.
5. Mandar mails con promociones y descuentos.

6. Funcionalidades mínimas

Su aplicación debe abarcar las siguientes funcionalidades mínimas:

- CRUD[1] de usuarios.
- CRUD de restaurantes.
- CRUD de platos.
- CRUD de órdenes.

[1] *Create, Read, Update and Delete*

- CRUD de comentarios.
- *Sign up* de usuario.
- *Log in* de usuario.
- Actualización de información de cuenta de usuario.
- Búsqueda de restaurantes.
- Búsqueda de platos.
- Votación de restaurantes y platos.
- Solicitud y respuesta de aplicación de restaurantes.

7. Requisitos mínimos de desarrollo

Para asegurar un producto de calidad, se les pide que utilicen las siguientes herramientas y buenas prácticas de desarrollo de *software*. Todas ellas son un estándar básico para la industria de software actual y potencian la producción de equipos de desarrollo.

7.1. *Kanban*: *Trello*

Utilizar el servicio de *Kanban Trello* para organizar su trabajo como equipo. Cada equipo debe tener un tablero de *Trello* que compartirá con su *product owner*. Éste puede tener la estructura (columnas) que el equipo encuentra conveniente mientras se note un claro flujo de trabajo que comunique el estado del proyecto a su *product owner*.

7.2. *Gitflow*

Para desarrollar la aplicación, gestionarán su proyecto en un repositorio *git*. Sobre esto, deben seguir el modelo de uso *Gitflow*. No es necesario seguirlo al pie de la letra, mientras se ocupen al menos dos *branches* principales y una *branch* por funcionalidad.

7.3. *Rubocop*

Seguir alguna guía de estilo de código para *Ruby* monitoreado por la gema *Rubocop*. Las configuraciones de estilo quedan a decisión de cada grupo, pero una vez fijadas deben respetarse.

7.4. *Heroku*

Utilizar la plataforma *Heroku* para publicar sus aplicaciones a producción.

7.5. *Docker*

Configurar sus ambientes de desarrollo con *Docker*.

8. Entregas, hitos y evaluación

El proyecto se llevará a cabo mediante desarrollo ágil inspirado en *Scrum*. Cada entrega se separa en un *Sprint* distinto, donde el trabajo para cada *Sprint* es negociado con su *product owner* en reuniones de *Sprint Review*.

8.1. *Product owner* y *Sprint Review*

Cada grupo de desarrollo tendrá asignado un *product owner* (ayudante) quien actúa como la contraparte del proyecto. Tras cada término de *Sprint* (entrega) se debe agendar una reunión (*Sprint Review*) con su *product owner* para discutir y monitorear el avance del proyecto. Además, deben definir junto a ella o él los pasos a seguir para el siguiente *Sprint*. **Todos los miembros del equipo deben asistir al *Sprint Review*** y debe planificarse para realizarse entre los **tres** inmediatamente siguientes días hábiles después del fin de un *Sprint*.

8.2. Coevaluación

Por cada entrega deberá responderse una coevaluación de sus compañeros. Ésta puede afectar positiva o negativamente su calificación. Detalles de ésta se especificarán luego de la primera entrega.

8.3. Evaluación

Su ayudante asignado es el encargado de evaluar su avance, además de llevar el rol de *product owner*. Para cada *Sprint Review*, su ayudante hará una sesión de corrección que dependerá de la entrega. Ésta puede implicar evaluación grupal y/o evaluación individual de conocimientos. Las notas parciales de cada entrega son **individuales** y consideran el avance grupal, individual y la coevaluación respondida.

8.4. Entregas

En total, son 4 entregas parciales. Cada entrega se realiza mediante su repositorio asignado de grupo en la organización de *GitHub* del curso, donde se corregirá el último *commit* en la rama *master* dentro de plazo. Luego de la entrega 0, todas incluyen avance de funcionalidades. Cuáles de ellas deben incluir en cada entrega depende de sus negociaciones con su *product owner*. Si bien este documento sirve como guía base de proyecto, **el resultado final puede (y debe) variar**. Adicionalmente, algunas entregas incluyen un aspecto obligatorio o evaluación específica a realizar. A continuación, se listan a grandes rasgos los entregables:

8.4.1. Entrega 0 (7 de Septiembre)

Relatos de usuario y aplicación mínima "Hello World!" publicada en *Heroku*.

8.4.2. Entrega 1 (27 de Septiembre)

Funcionalidades, modelación mediante diagrama E/R de la aplicación, evaluación individual de conocimientos sobre *Docker*.

8.4.3. Entrega 2 (18 de Octubre)

Funcionalidades y evaluación individual de conocimientos sobre *Ruby on Rails*.

8.4.4. Entrega 3 (8 de Noviembre)

Funcionalidades y segunda evaluación individual de conocimientos sobre *Ruby on Rails* o *Docker* como oportunidad de corrección de nota.

8.4.5. Entrega 4 (27 de Noviembre)

Funcionalidades faltantes e integración de una *API* a elección. Implementación completa de la aplicación web.

8.5. Presentación final (TBA)

Finalmente, luego de las entregas parciales se realizará una presentación del producto logrado al equipo docente del curso. En esta oportunidad se busca que el equipo de desarrollo **completo** presente lo experimentado durante el desarrollo, el resultado obtenido y las lecciones aprendidas.

8.6. Nota

Como se especifica en el programa del curso, la nota de proyecto se divide en tres componentes:

- $\overline{E}_P$: Promedio de notas de entregas parciales.
- E_F: Nota de entrega final, como producto desarrollado.
- P_F: Nota de presentación final.

La nota de proyecto (P) se calcula como sigue:

$$P = 0.5 \cdot \overline{E}_P + 0.2 \cdot E_F + 0.3 \cdot P_F$$

9. Recomendaciones Finales

Tienen bastante libertad para construir una aplicación de acuerdo a lo que cada grupo considere que es importante dado el espíritu y los objetivos que se explicaron al comienzo. Las funcionalidades (features) deben ser negociadas con el *product owner* (el ayudante que les seguirá durante todo el semestre) Se recomienda identificar y levantar la mayor cantidad posible de épicas y relatos de usuario en la *Entrega o Sprint 0* a pesar que no terminen siendo todos finalmente implementados.

10. Política de integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Academica publicado por la Dirección de Docencia de la Escuela de Ingeniería en el SIDING.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1.1 en el curso y se solicitará a la Dirección de Pregrado de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente. Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.