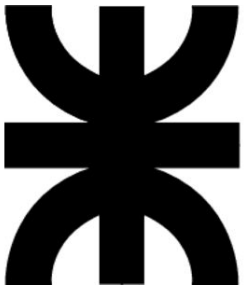


Universidad Tecnológica Nacional
Facultad Regional Córdoba
Ingeniería en Sistemas de información

Cátedra: Ingeniería de Software

Trabajo Práctico: “REQUERIMIENTOS ÁGILES - Implementación de user stories”

Curso: 4K2

Grupo n°: 9

- 76034 Mauricio Zapata
- 74973 Isaías Avalle Borgiani
- 77629 Francisco Maurino
- 77020 Rodrigo Diaz Mac William
- 64797 Mariana Fagandini

Docentes:

- Adjunto: Meles, Silvia Judith
- JTP: Massano, María Cecilia
- Ayudante 1ro: Robles, Joaquín Leonel

Índice

Índice **2**

Enunciado **3**

Desarrollo **5**

Documento de estilos 5

Estructura de archivos 5

Nombres claros 7

Regla de los 5 segundos 8

Organizar el código 8

Proveer claridad 9

Componentes 9

Servicios 10

Complementos: Angular Material y Bootstrap 10

Link implementación de US 11

Anexo de preguntas 12

Listado de Preguntas del grupo 12

Preguntas de otros grupos 13

Enunciado

Unidad:	Unidad Nro. 3: Gestión del Software como producto
Consigna:	Implementar una User Story determinada usando un lenguaje de programación elegido por el grupo respetando un documento de reglas de estilo.
Objetivo:	Que el estudiante comprenda la implementación de una User Story como una porción transversal de funcionalidad que requiere la colaboración de un equipo multidisciplinario.
Propósito:	Familiarizarse con los conceptos de requerimientos ágiles y en particular con User Stories en conjunto con la aplicación de las actividades de SCM correspondientes.
Entradas:	Conceptos teóricos sobre el tema desarrollados en clase. Definición completa de las User Stories correspondientes al Trabajo Práctico 2 “Requerimientos Ágiles – User Stories y Estimaciones”
Salida:	- Implementación de la User Story correspondiente en un programa ejecutable - Documento de estilo de código

Instrucciones:	- Seleccionar una User Story a implementar de entre las siguientes opciones:
	  - *Realizar Pedido a Comercio adherido* (grupos pares)
	  - *Realizar un Pedido de "lo que sea"* (grupos impares)
	- Seleccionar el conjunto de tecnologías para implementar la funcionalidad elegida.
	- Buscar y seleccionar un documento de buenas prácticas y/o reglas de estilo de código para el lenguaje de programación a utilizar.
	- Implementar la US siguiendo las reglas de estilo determinadas.

User story:

Realizar Pedido de "lo que sea" Como Solicitante quiero realizar un Pedido de "lo que sea" para recibir algo en mi domicilio que no está disponible en los comercios adheridos **Nota**: Se debe indicar qué debe buscar el Cadete con un campo de texto **Nota**: Se puede adjuntar opcionalmente una foto en formato JPG con un tamaño máximo de 5 MB. **Nota**: Se debe indicar la dirección del comercio en forma textual (calle, número, ciudad y referencia opcional en formato de texto) o seleccionando un punto en un mapa interactivo de Google Maps. **Nota**: Se debe indicar la dirección de entrega (calle, número, ciudad y referencia opcional en formato de texto). La ciudad podrá ser seleccionada de un listado de Ciudades disponibles. **Nota**: Se debe seleccionar la forma de pago: Efectivo o Tarjeta VISA, en caso de haber seleccionado pago en Efectivo el monto con el que va a pagar. En caso de seleccionar Tarjeta VISA debe ingresar el número de la tarjeta, nombre y apellido del Titular, fecha de vencimiento (MM/AAAA) y CVC **Nota**: Debe ingresar cuando quiere recibirlo: "Lo antes posible" o una fecha/hora de recepción)	8

- Probar realizar un Pedido de "lo que sea" en efectivo "lo antes posible" (pasa)
- Probar realizar un Pedido de "lo que sea" con tarjeta "lo antes posible" (pasa)
- Probar realizar un Pedido de "lo que sea" programando la fecha/hora de entrega (pasa)
- Probar realizar un Pedido de "lo que sea" con una tarjeta inválida (falla)
- Probar realizar un Pedido de "lo que sea" con una tarjeta MasterCard (falla)
- Probar realizar un Pedido de "lo que sea" en efectivo sin indicar el monto a pagar (falla)
- Probar realizar un Pedido de "lo que sea" programando una fecha/hora de entrega no válida (falla)
- Probar realizar un Pedido de "lo que sea" sin especificar qué buscar (falla)
- Probar realizar un Pedido de "lo que sea" adjuntando una foto (pasa)
- Probar realizar un Pedido de "lo que sea" sin indicar la dirección del comercio (falla)
- Probar realizar un Pedido de "lo que sea" seleccionando la dirección del comercio en el mapa interactivo (pasa)

Desarrollo

Documento de estilos

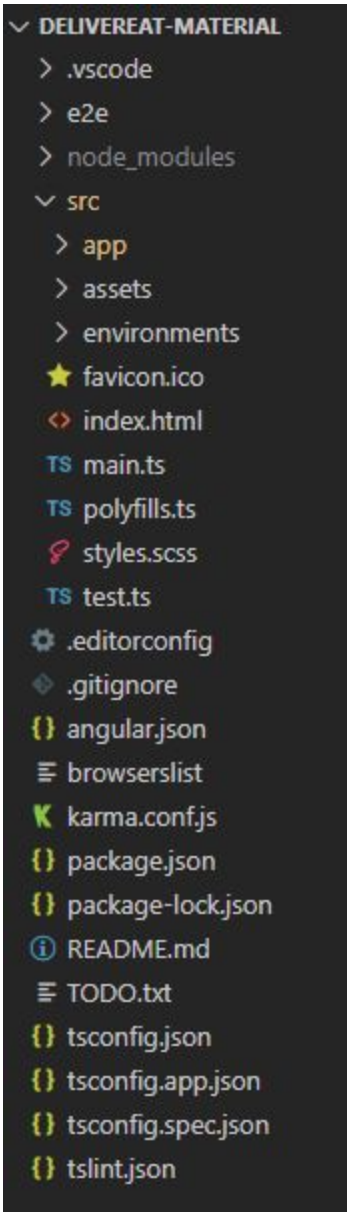
Antes de comenzar a leer este documento se recomienda ver la guía de estilos oficial de angular en:

https://angular.io/guide/styleguide

donde se describe de forma muy completa y actualizada como trabajar con Angular. Se usará Angular 8 para el desarrollo de la implementación de la US.

Estructura de archivos

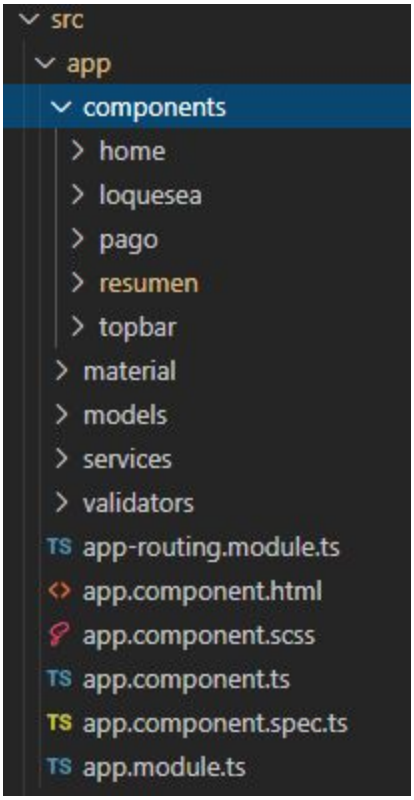
Angular CLI nos provee una forma de crear un nuevo proyecto con una estructura bastante cómoda y práctica:



Generalmente se define la estructura de acuerdo a la complejidad del proyecto. Entre más grande sea, más orden y modularidad requiere.

Una de las mayores ventajas que proporciona angular frente a otros frameworks, es principalmente la capacidad de mantener ordenados proyectos complejos, haciendo uso de la excelente modularidad que presenta. A pesar de que el presente proyecto no es de gran envergadura, sigue siendo de gran utilidad la estructura proporcionada por el framework a la hora de organizar las carpetas del proyecto. Se logra en primer lugar una separación de intereses haciendo uso de componentes, los cuales a su vez, internamente proporcionan una clara separación entre la capa visual (la vista), la lógica de negocio (controlador), y los datos

(modelo). Tal modularidad permite una mayor facilidad en la gestión y mantenimiento del proyecto, haciendo posible la labor conjunta de un equipo de desarrollo de un modo más eficaz.



Tener una estructura bien definida nos ayudará a pensar en escalabilidad y permitirá ubicar fácilmente los distintos archivos a medida que el proyecto crezca.

Nombres claros

Uno de los mayores problemas a la hora de entender el código, suele ser el nombre que se le asigna a los métodos, variables o parámetros. Mientras más claro sea uno a la hora de definir los nombres mientras escribe el código, mejor será su entendimiento a futuro o para alguien que intente leerlo y comprenderlo posteriormente facilitando el mantenimiento del mismo.

Si nos fijamos en el ejemplo siguiente podemos ver lo difícil que es entender el objetivo del método en el primer caso. En cambio si tenemos en cuenta lo enunciado en el párrafo anterior, podemos ver que la diferencia es significativa, a pesar de que se trata del mismo método:

```
getU(n: string): any[] {
  return this.u.filter(user => {
    return user.n === n;
  });
}
```

```
getUsersByName(userName: string): User[] {
  return this.users.filter(user => {
    return user.name === userName;
  });
}
```

Regla de los 5 segundos

Podemos aplicar la regla conocida como “5 segundos” a nuestro código. Si nos toma más de 5 segundos entender un bloque de código, es mejor considerar refactorizar.

La idea es que el código se entienda en el menor tiempo posible. Eso se puede lograr creando diferentes funciones pequeñas y haciendo composición en funciones más complejas. En caso de que el código falle, es más fácil encontrar el error y corregirlo sin afectar otras funciones.

Organizar el código

Algunas formas de tener un archivo de código más organizado y legible son:

- Lo más importante debe ir arriba.
- Primero propiedades, después métodos.
- **Un Ítem para un archivo**: cada archivo debería contener solamente un componente, al igual que los servicios.
- **Solo una responsabilidad**: Cada clase o módulo debería tener solamente una responsabilidad.

- **El nombre correcto**: las propiedades y métodos deberían usar el sistema de camel case (ej: getUserByName), al contrario, las clases (componentes, servicios, etc) deben usar upper camel case (ej: UserComponent).
- Los componentes y servicios deben tener su respectivo sufijo: UserComponent, UserService.
- **Imports**: los archivos externos van primero.

Proveer claridad

Al escribir código, es probable que alguien más lea el mismo en algún momento. Es por eso que lo ideal cuando escribimos es pensar en la persona que lo leerá, o incluso en ti mismo.

Código auto-descriptivo:

- Explicar en el mismo código, no en comentarios.
- Los comentarios deben ser claros y entendibles.

Evita comentar si:

1. Se trata de explicar qué hace el código (dejar que este sea tan claro que se explique solo).
2. Se tiene funciones y métodos bien nombrados. No es necesario ser repetitivo.

Comentar cuando:

1. Se trate de explicar por qué se hizo lo que se hizo.
2. Se trate de explicar las consecuencias del código escrito.

Componentes

En Angular, lo más importante debe ir al inicio. Dentro de los componentes generalmente se escribe primero las propiedades y luego los métodos. Así mismo, a veces suelen agruparse propiedades o funciones alfabéticamente, mientras que en otros casos se ordenan por funcionalidad. Lo importante aquí es mantener una consistencia durante todo el proyecto. Además:

- ➢ Es importante tratar de escribir código lo más compacto posible. Cada quien tiene una forma distinta de escribir y estructurar sus funciones.

- ➢ Es deseable que los componentes se mantengan lo más simple posible. En este contexto, se delega la mayor parte de la lógica a los servicios.

- ➢ Mantener la consistencia a la hora de declarar funciones, evitar las faltas ortográficas y mantener la seriedad de los nombres utilizados.

Servicios

Algunas de las mejores prácticas a la hora de crear servicios son:

- Crear los servicios como Injectables para lograr un código con menor acoplamiento

- Utilizar servicios para recolectar datos: es total responsabilidad de los servicios recolectar la información necesaria, ya sea haciendo uso de una API, localstorage o alguna estructura haya sido creada por el desarrollador. Los componentes nunca deben encargarse de resolver como traer información, estos sólo deberían encargarse de llamar al servicio que contiene todo lo necesario.

Priorizar las buenas prácticas a la hora de escribir código rinde sus frutos en el futuro, logrando un código mucho más mantenible.

Complementos: Angular Material y Bootstrap

Para la implementación de la US se utilizará Angular Material. Se trata de una librería de estilos basada en la guía de diseño de Material Design. Resulta bastante útil para presentar la información de las aplicaciones con una vista y un diseño estéticamente correctos y amigables.

Por otro lado también se hará uso de la librería Bootstrap, la cual resulta de gran utilidad a la hora de implementar plantillas, formularios y diversos elementos visuales en HTML y CSS. Provee clases predefinidas, las cuales permiten ahorrar gran cantidad de esfuerzo al momento de implementar el front-end y son de tipo responsive, lo que facilita el desarrollo de diversas aplicaciones y resulta óptimo para el desarrollo mobile en cuestión.

Basado en:

Guia de estilo - Angular

Angular Material

Link implementación de US

Maquetado inicial: Maquetado inicial en Figma

Repositorio: GitHub DeliverEat

Anexo de preguntas

Listado de Preguntas del grupo

- Es necesario hacer pantalla con Botón que lleve a “pedi lo que quieras” (pantalla previa) o directamente la interfaz del pedido?: Pongan una pantalla previa
- La opción “Lo antes posible” bloquea la selección de fecha y hora? Bloquea la selección de fecha
- La dirección de entrega también puede seleccionarse en un mapa interactivo? Si
- La dirección del comercio también tiene un combo con ciudades disponibles? Si
- ¿Qué pasa si el comercio se encuentra en la ciudad A y la dirección de entrega es de la ciudad B? No se puede seleccionar el comercio
- ¿Sería correcto tomar como ciudad de entrega la que se seleccionó previamente a la hora de indicar el comercio e impedir modificar la misma? ¿O simplemente se presenta un mensaje indicando que se debe cambiar la ciudad de entrega o cancelar el pedido? Es correcta la primera opción
- ¿Hay algún color definido para las pantallas de la aplicación? Tonos azules
- ¿Solo se adjunta una foto descriptiva de producto? ¿Si son más, 5 MB máx por foto o en total? 1 imagen de 5 MB x imagen máximo
- ¿Se muestra el mapa interactivo y campos a completar de direcciones (ciudad calle numero referencia) al mismo tiempo o debe ir en pantallas diferentes? En la misma pantalla pero se usa una opción o la otra
- El costo de pedido es fijo $70 según una respuesta del profe Joaquin. ¿Ese es el importe total, o es lo que se cobra extra a lo que salga el pedido? Si es un extra, ¿cómo se calcularía? Porque no hay ningún comercio ni precio registrado previamente, entonces no sabemos cómo se podría deducir dicho costo. Tomen solo ese costo fijo.

  Documento de estilo de Código:

- ¿Sacamos de internet o de documentación de alguno de los lenguajes? Investiguen y seleccionen el que consideren mejor
- ¿Qué es exactamente lo que debería describir dicho documento? Es parte de lo que deben investigar.
- En la tarea de la uv, ¿subiriamos el documento de estilos únicamente y la implementación la mostramos en vivo, o también subimos todo el proyecto? Suben solo el documento de estilo

Preguntas de otros grupos

- ¿Qué ciudades se van a incluir en el mapa?: Todas las ciudades de la provincia de cba(incluyan una muestra de 10 ciudades)
- ¿Qué ciudades se van a incluir en el listado de ciudades disponibles?: lo mismo que arriba
- ¿La tarjeta visa es credito o debito?: Credito
- ¿Como es el formato de fecha/hora? dd/mm/aaaa
- ¿Se debe realizar un detalle del pedido cuando se confirma el mismo?: si
- ¿Es necesario mostrar el vuelto al cliente antes de que confirme el pedido? : si
- Si se ingresa un dato erróneo, se debe controlar a medida que se van llenando los campos? O cuando se hace click en el botón que registra el pedido? cuando el campo pierde el foco
- La pantalla previa a la del pedido de "lo que sea", que botones tiene que tener? no aplica a la US
- Al usar la API de Google Maps, en el mapa queda el siguiente mensaje "For development purposes only". ¿Descuentan puntos por eso? no
- Si la dirección de entrega/comercio se indica de forma textual, se bloquea la opción de selección por el mapa? debe poder corregir la ubicación del pin
- ¿Alguno de los siguientes campos tiene límite de caracteres? Campo de texto donde se indica que tiene que buscar el cadete, campo de texto "Referencia", calle, número, nombre y apellido del titular de la tarjeta. todos soportan hasta 255 caracteres
- Cuando pedís lo que sea, ¿se puede pedir más de 1 cosa en el pedido? Similar a pedir con carrito en la otra historia absolutamente mientras entre en la mochila
- ¿Tenemos que hacer categorías de comercio (para filtrar) como en la otra historia? no aplica a la US
- ¿Hay que poner un campo para que el usuario coloque el monto de recompensa por el servicio del cadete o se cobra un porcentaje del precio para el cadete? De ser así, ¿cómo se calcula? el costo de envío es fijo en $70
- ¿Hay fuentes y tamaños de letra definidos? Roboto + OpenSans (tamaño a determinar por uds) https://fonts.google.com/specimen/Roboto#about
- El logo y el nombre de la app tiene que figurar en la pantalla del pedido "lo que sea"? De ser así, de donde obtenemos el logo? Si, el logo puede estar representado con cualquier ilustración de una moto vespa