Interrogación 2: Ingeniería de Software
Semestre 2- 2022

Nombre Completo: _____

Sección (profesor): _____

Pregunta 1 (16 puntos)

- Cada inciso vale 2 puntos.
- La repuesta no tiene que se exactamente igual a la pauta siempre y cuando la respuesta vaya en la misma dirección y demuestre que el estudiante conoce el concepto relacionado a la pregunta.
- Es posible asignar la mitad del puntaje (1 punto), en casos donde la respuesta no este al 100%. Por ejemplo, solo se describio 2 ventajas de las 3, o solo se explico acoplamiento de forma correcta y no cohesion, etc.

Responda en no más de *5 líneas* las siguientes preguntas:

- ¿Cuáles son los problemas asociados a la sobre estimación y a la subestimación?
  - Subestimar genera un enorme nivel de estrés en el equipo que puede traducirse en una baja de rendimiento general sin contar problemas de calidad o el riesgo de perder a parte del equipo a medio camino.
  - El sobreestimar el tiempo de desarrollo está asociado a dos problemas conocidos como la ley de Parkinson y la ley de Goldratt. E*l trabajo se expandirá hasta usar todo el tiempo disponible. Si hay demasiado tiempo se va a malgastar el tiempo al comienzo.*

- El desarrollo de un software de 200 story points tomó 6 meses. Ahora el mismo equipo se propone iniciar un proyecto de 400 sps que está planificado para 12 meses. ¿Es eso razonable? Explique.
  - En cuanto al supuesto de crecimiento lineal del esfuerzo en relación al tamaño, para tamaños relativamente pequeños no se induce un error demasiado grande. 12 meses es un periodo largo lo que involucra un error de estimación considerable. Por lo que se considera no razonable. En particular, porque no se sabe las características del nuevo proyecto, que seria una fuente de incertidumbre.

- ¿Qué se entiende por "cono de incertidumbre"? Explique brevemente.
  - A medida que el proyecto avanza la incertidumbre disminuye generando una curva que se conoce como el *cono de incertidumbre.* El cono de la incertidumbre muestra que los errores en las estimaciones disminuyen a medida que el proyecto avanza.

- Escriba un ejemplo de una clase que tiene poca cohesión. Explique su respuesta.
  - Una clase con poca cohesión sería una clase donde sus métodos no están relacionados con sus atributos o con otros métodos. Por ejemplo, en esta clase el método foo no utiliza ninguno de los atributos de la clase.

```
class A{
    att_reader: :a
    att_reader: :b
    foo(){  puts 'hola'}
}
```

  - Con contar las lineas ocupadas para escribir el código de ejemplo dentro de las 5 lineas mencionadas en el ennciado.
  - Debería ser suficiente explicar que una clase con poca cohesión es una en que hay métodos que no se relacionan con atributos de la clase

- ¿Por qué es deseable un diseño con alta cohesión y bajo acoplamiento?
  - Un bajo nivel de acoplamiento va a incidir en que el sistema sea más fácil de entender, modificar, extender y testear.
  - La cohesión tiene que ver con que tanto tienen que ver entre sí las cosas que decidimos dejar juntas en una misma unidad. Una clase enfocada en una sola cosa es más fácil de entender, mantener y extender.

- ¿Cuál es la relación entre "tier" y "layer"?
  - Un layer es una abstracción lógica. Por ejemplo la capa de transporte en el modelo OSI de redes. Un tier involucra una unidad física (por ejemplo el cliente o el servidor). Es posible implementar las capas en una dos o más tiers.
  - Complementariamente a lo anterior, un tier puede ser un computador o un proceso independiente. Por ejemplo, yo puedo correr el cliente y el servidor en la misma máquina pero están en proceso diferentes. Las capas (layer) agrupaciones lógicas de componentes, por ejemplo, capa de persistencia, aplication, UI, etc. Varios layers pueden sera parte de un mismo tier.
  - No tiene que decir explícitamente esto, pero si dan a entender que entienden la diferencia.

- Indique 3 ventajas y 3 desventajas de una arquitectura de microservicios en comparación a una arquitectura monolítica.
  - Ventajas
    - Es posible desarrollar y deployar en forma independiente cada componente (servicio)
    - Cada componente puede tener su propia tecnología y su propia base de datos
    - Si es necesario escalar no es necesario hacer copias de la aplicación completa (solo de los servicios más requeridos)
    - Es posible parar un servicio sin necesidad de parar toda la aplicación
    - Es más fácil reemplazar un servicio, incluso con la aplicación en funcionamiento
  - Desventajas
    - Mayor costo de desarrollo para aplicaciones pequeñas y medianas
    - Mayor complejidad para la mantención y manejo de versiones
    - Dificultad para asegurar SLAs
    - Depende mucho más de la red

- Ojo el enunciado solo pide 3 de cada una.

- Indique 2 diferencias entre la arquitectura clásica de servicios (SOA) con la arquitectura de microservicios.
  - Los microservicios son más pequeños, los servicios en SOA pueden ser procesos de negocio completos
  - No se requiere una capa de middleware intermedia entre los consumidores de los servicios y los servicios mismos (la API Gateway es opcional)
  - La granularidad de los servicios es mucho menor. Esto facilita la reutilización de estas componentes ya que al ser demasiado grandes muchas veces no servían para incorporarlos en más de una aplicación.
  - Desaparece la necesidad del bus de servicios y por lo tanto de un middleware que permita conectar o comunicar los servicios
  - Tienden a usar el protocolo REST en lugar de SOAP para interactuar

Pregunta 2 (20 puntos)

A continuación se presenta el código Ruby que sirve para implementar un sistema de facturación. El objeto de tipo *PaymentSystem* mantiene una lista de las órdenes y de los productos que se manejan. La orden incluye una serie de items (uno por cada producto comprado) que deben ser impreso uno por línea.

A. (10 pts) Dibuje el diagrama de clases que corresponde al código dado (incluya todo lo que puede obtenerse a partir del código)

B. (10 pts) Dibuje un diagrama de secuencia que muestre como se lleva a cabo la impresión de una factura cuando el objeto de tipo RetailOrder recibe un mensaje printOrder.

```ruby
class PaymentSystem
   @@orders
   @@products
   def self.process
       orders.each do |order|
           order.printOrder
       end
   end
   def self.getProduct (code)
       products.each |prod|
           if code == prod.code
                return prod
           end
   end
   def self.addProduct(product)
       products << product
   end
   def self.addOrder(order)
       orders << order end
   end
end
```

```ruby
class Product
   attr_reader :code, :name, :price
   def initialize (code, name, price)
       @code = code
       @name = name
       @price = price
   end
end
```

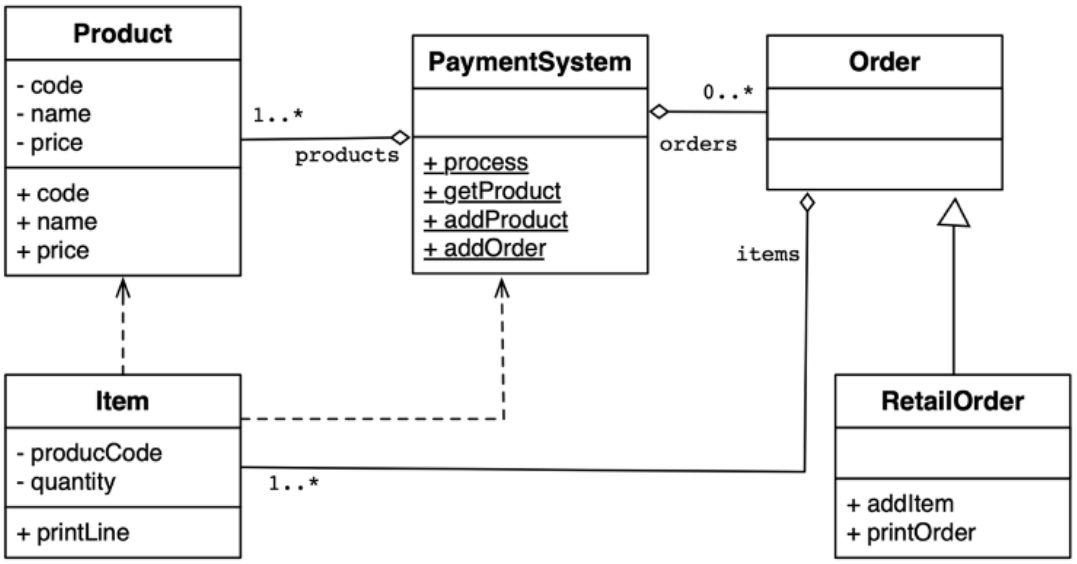
```ruby
class Item
   @productCode
   @quantity
   def printLine
    product = PaymentSystem.getProduct(@productCode)
    amount = product.price * quantity
    puts "#{quantity} #{product.code}
          #{product.name} #{product.price} #{amount}"
   end
end
```

```ruby
class Order
   @items
end
```

```ruby
class RetailOrder < Order
   def addItem (code, quantity)
       theItem = Item.new (productCode, quantity)
       @items << item
   end
   def printOrder
       items.each do |item|
           item.printLine
       end
   end
end
```

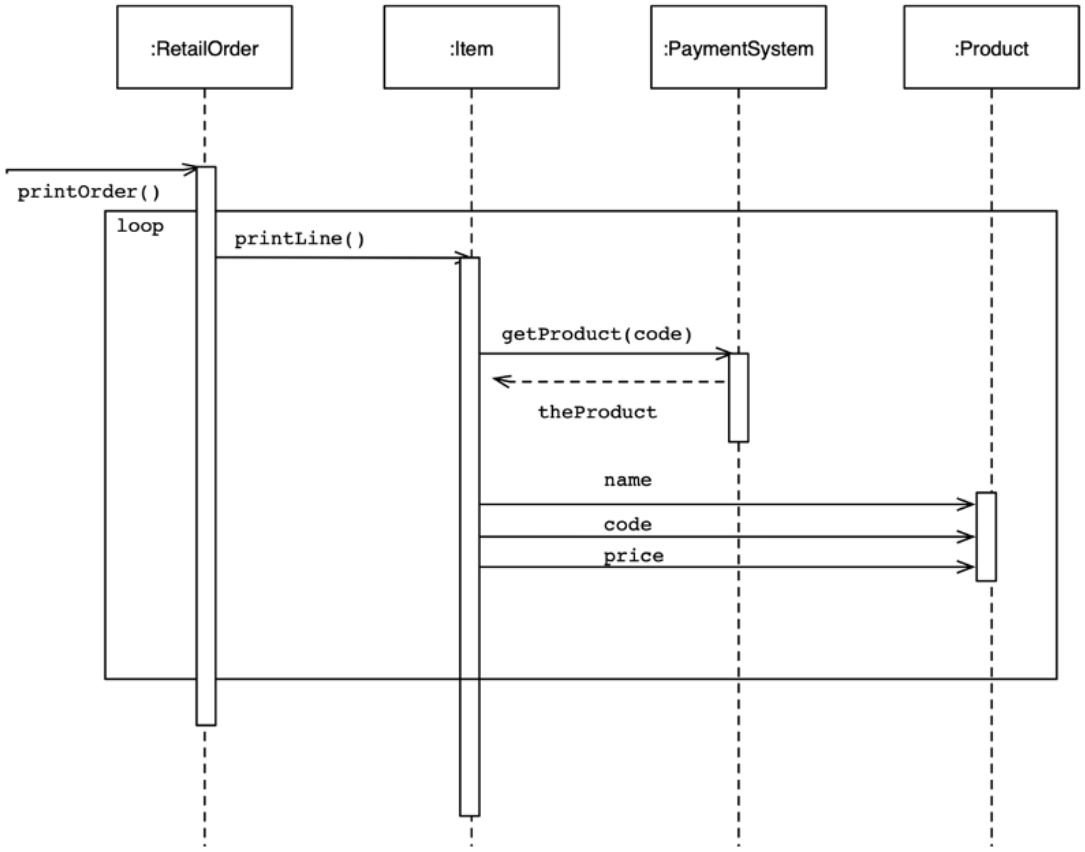
Solución

a)



- 1 punto por clase
  - nombre, atributos, metodos
- 1 punto por relación entre clase
  - revisar que la relación este correcta (linea punteada, rombo, linea recta, herencia)
- no descontar por cardinalidad (1 .. *)
- no descontar si no subrayo los nombres de los métodos de la clase PaymentSystem
- si suma mas de 10, solo poner 10.

b)



Nota:

- Descontar un punto o medio-punto a criterio si es que se olvidaron un detalle importante en el diagrama, por ejemplo:
    - No dibujaron una llamada un método (printLine, getProdcut, name, code, price)
        - o dibujaron la linea sin poner el nombre del metodo encima
    - No dibujaron la linea punteada (the product) , el valor retornado.
    - No dibujaron el objeto order, item o payment system, o producto
        - En caso de los objetos items, podrian haber 1 o mas, ya que no se especifico cuantos objetos items forman parte de la orden.
- Si no aparece el rectángulo correspondiente al loop o está indicado de otra forma OK.

Pregunta 3 (24 puntos)

El Dr Nefario está creando un nuevo juego de cartas para jugar con los minions. Todavía no lo tiene definido al 100%, pero de algo está seguro, que durante una partida (`Game`) se repartirán las cartas (`Card`) de un mismo mazo (`Deck`) a N jugadores de diferente forma a lo largo del juego. La estrategia de repartición de cartas podrá cambiar a lo largo de una misma partida. Existen tres formas de repartir las cartas:

- **En bloque**, donde la computadora retira 5 cartas del mazo y se las entrega al primer jugador. Posteriormente, retira otras 5 cartas del mazo y se las entrega al segundo jugador, y así sucesivamente hasta que todos los jugadores hayan recibido 5 cartas. Se reparten las cartas en el orden en que los jugadores aparecen en la lista (@players)
- **Uno por uno,** la computadora retira una carta y la entrega al primer jugador. Posteriormente, retira otra carta y se la entrega al segundo jugador y así sucesivamente hasta que todos hayan recibido una carta. Se reparten las cartas en el orden en que los jugadores aparecen en la lista (@players)
- **Solo uno,** donde la computadora retira una sola carta y se la entrega aleatoriamente a un jugador. Los demás jugadores no reciben ninguna carta.

El Dr. Nefario realizó un pequeño modelo orientado a objeto y el método *split* que permite repartir las cartas **en bloque** a N jugadores. El Dr. Nefario se enteró que en el curso de Ing. de Software se estudian patrones de diseño, por lo que decidió delegar la tarea de implementar las otras formas de repartir cartas a los estudiantes del curso. Su única condición es que modifiquen su clase ***Game*** y utilicen el patrón ***strategy*** de forma tal que el pueda agregar nuevas formas de repartir cartas en el futuro de forma fácil, iterativa e incremental (**extensible!**).

A. (16 pts) Implemente el patrón ***strategy***. Modifique la clase Game, y agregue las clases necesarias para implementar el patrón ***strategy***. Su solución debe implementar las tres estrategias de repartición escritas: en bloque, uno por uno, solo uno.

B. (8 pts) Escriba un pequeño código de ejemplo donde se repartan cartas a 3 jugadores, de la siguiente forma: (i) primero se reparten 5 cartas a cada jugador (**en bloque**), (ii), se reparte 1 carta a cada jugador (**uno por uno**); (iii) se repare una carta a un jugador al azar (**sólo uno**). Note que todas las reparticiones se deben hacer con el mismo mazo (*Deck*) en la misma partida (Game).

Solución A.

```ruby
class Splitter
   def split(deck, players)
       raise NotImplementedError
   end
end
```

```ruby
class BlockSplitter
   def split(deck, players)
       players.each do  |player|
           5.times{
           player.add(deck.pickLast)
           }
       end
   end
end
```

```ruby
class OneAllSplitter
   def split(deck, players)
       players.each do  |player|
           player.add(deck.pickLast)
       end
   end
end
```

```ruby
class OnlyOneRandomSplitter
   def split(deck, players)
       players.sample.add(deck.pickRandom)
   end
end
```

```ruby
class Game
   def split(splitter)
       splitter.split(@deck,@players)
   end
end
```

La solución puede variar un poco, no descontar puntajes por detalles de sintaxis. Si se escribió código en otro lenguaje no dar puntaje.

- 4 puntos, por modificar laa clase game para que pueda cambiar y ejecutar estrategias. Puede ser en un solo metodo como la solución anterior o dos, por ejemplo, con un set strategy y luego un apply strategy. U otra solución permita aplicar la estrategia.
- 4 puntos por estrategia implementada correctamente. Descontar puntos a criterio, si falto un detalle importante en la implementación.

Solución B.

```ruby
game = Game.new(3)
game.shuffle() # esta lina es opcional
game.split(BlockSplitter.new)
game.split(OneAllSplitter.new)
game.split(OnlyOneRandomSplitter.new)
```

Puede variar la implementación un poco, pero la idea es que las estrategias cambian en tiempo de ejecución, y sean aplicadas al mismo objeto game. De esta forma se reparte utilizando el mismo mazo. Descontar puntos a criterio, por ejemplo, si no aplico las estrategias al mismo objeto.

Pregunta 4 (20 puntos)

Considere el siguiente código de un estudiante de Programación Avanzada (kokito) que está iniciandose en programación orientada a objetos. El ha modelado posteos (objetos posts) en un muro (objeto wall). De momento el código es simple y considera 3 tipos de posts: simple post, image post y check-in post. Para realizar pruebas de su modelo orientado a objetos el estudiante ha creado el siguiente script en ruby:

ruby main.rb

```
wall = Wall.new
wall.add(Post.new("quiz on Thursday",
                  "Juan P."))
imagePost = ImagePost.new("Me and Oscar Niestraz",
                          "Juan P.",
                          "/users/juan/picturas/oscar_with_juan.jpg")
imagePost.like()
imagePost.like()
wall.add(imagePost)
checkingPost = CheckInPost.new("I really like Thai Food",
                               "Juan P.",
                               "Tao Pai Pai restaurante")
checkingPost.addComment('a really nice place!')
checkingPost.addComment('such a nice restaurant')
wall.add(checkingPost)
wall.print
```

El código anterior imprime el siguiente texto en la consola. Cada post imprime su tipo. El post con imagen y el check-in post imprimen la foto y el lugar del post respectivamente.

Console Output

```
-= Simple Post =-
Juan P. -- quiz on Thursday
0 likes
-= Image Post =-
Juan P. -- Me and Oscar Niestraz
Photo: [/users/juan/picturas/oscar_with_juan.jpg]
2 likes
-= CheckIn Post =-
Juan P. -- I really like Thai Food
Checking at Tao Pai Pai restaurante
0 likes
comments:
   a really nice place!
   such a nice restaurant
```

El código funciona como esperaba kokito. Sin embargo, kokito se ha dado cuenta que su código tiene código duplicado en diferentes clases y está preocupado porque esto puede dificultar su mantenimiento. Su misión es **modificar el código de kokito para eliminar el código duplicado aplicando el patrón template method**. El código después de aplicar el patrón no debe tener ninguna línea de código duplicada y el archivo main.rb debe imprimir el mismo texto en la consola.

Solución

```ruby
class Post
   def initialize(msg,auth)
       @message = msg
       @author = auth
       @likes = 0
       @comments = []
   end
   def print
       printPostKind
       printAuthorMessage
       printBody
       printLikes
       printComments
   end
   def like()
       @likes = @likes + 1
   end
   def addComment(comment)
       @comments.push(comment)
   end
   def printPostKind
       puts "-= Simple Post =-"
   end
   def printAuthorMessage
       puts "#{@author} -- #{@message}"
   end
   def printLikes
       puts "#{@likes.to_s} likes"
   end
   def printComments
       if @comments.length > 0
           puts "comments:"
           @comments.each do |each|
               puts "    #{each}"
           end
       end
   end
   def printBody

   end
end
```

```ruby
class ImagePost < Post
   def initialize(msg,auth,filePath)
       super(msg,auth)
       @filePath = filePath
   end
   def printPostKind
       puts "-= Image Post =-"
   end
   def printBody
       puts "Photo: [#{@filePath}]"
   end
end
```

```ruby
class CheckInPost < Post
   def initialize(msg,auth,placeName)
       super(msg,auth)
       @placeName = placeName
   end
   def printPostKind
       puts "-= CheckIn Post =-"
   end
   def printBody
       puts "Checking at #{@placeName}"
   end
end
```

- Debe estar correctamente aplicado el patron template, es decir, debe existir un metodo en la clase padre que llame a otros metodos dentro de la misma clase. Las clases hijas deben sobre escribir uno o mas de estos metodos.
    - Si no aplico patron no tiene ningun puntaje.

- Si aplico el patron
    - Descontar 4 puntos por linea de código duplicada en la solución.

Pregunta 5 (20 puntos)

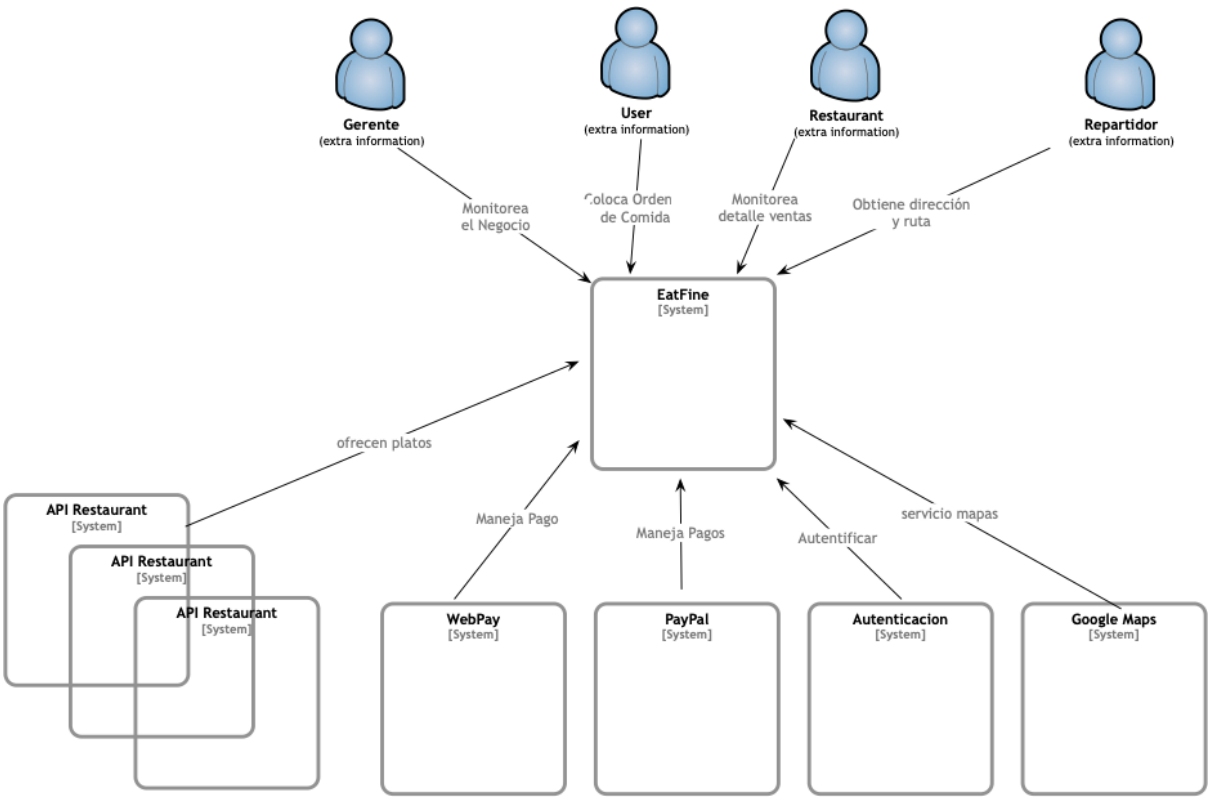
Supón que estás enfrentado a diseñar una aplicación ***EatFine*** de pedidos de comida a restaurantes desde cero con las siguientes características

- la app debe estar disponible en smartphones, tablets y laptops
- los usuarios de la app pueden ver platos ofrecidos por diversos restaurantes y ordenarlos desde sus casas para entrega a domicilio
- los restaurantes que quieren enrolarse no se les cobra nada de base pero deben proveer una API para acceder a su oferta e interactuar con la aplicación
- los usuarios pueden autenticarse con un servicio externo
- los usuarios pueden pagar solo con WebPay o PayPal
- la app utiliza los servicios de Google Maps para ayudar a buscar restaurants en un área
- los repartidores (motoristas) deben recoger la comida y llevarla a su destino (la app debe ayudar en esto también)
- se les envía una vez al mes a cada restaurante adherido el dinero recaudado de la venta de sus platos menos un descuento de 5% por servicio
- los restaurantes adheridos pueden monitorear el detalle de sus ventas a través de la app para revisar el detalle de las plantas

Se pide hacer un primer borrador de la arquitectura de este sistema y expresarlo en la forma del modelo C4. Solo se piden los dos primeros niveles: ***Contexto*** y ***Containers***.

Solución

- Pueden existir varias posibles soluciones. Lo importante es que el estudiante entienda bien que elementos deben aparecer en cada nivel.
    - El de contexto, muestra el usuario, el sistema (EatFine), y los subsistemas grandes que interactúan con el. Deben aparecer los subsistemas menciandos en el enunciado. Por ejemplo, WebPay, Google Maps.
    - El de Container, debe mostrar los subsistemas que dan forma al sistema EatFine. En base al Texto anterior. Por ejemplo, Web App, Mobile , iPad, database, API Aplications.
- Descontar puntaje a criterio segun el comentario anterior. Por ejemplo, si un sistema aparece en el diagrama de contexto y debería aparecer en el containers. O si falta un subsistema importante mencionado en el texto. Por ejemplo, si no pusieron Google Maps, entre otras cosas.


Gerente
(extra information)
User
(extra information)
Restaurant
(extra information)
Repartidor
(extra information)
Monitorea el Negocio
Coloca Orden de Comida
Monitorea detalle ventas
Obtiene dirección y ruta
EatFine
[System]
ofrecen platos
Maneja Pago
Maneja Pagos
Autentificar
servicio mapas
API Restaurant
[System]
API Restaurant
[System]
API Restaurant
[System]
WebPay
[System]
PayPal
[System]
Autenticacion
[System]
Google Maps
[System]

Nivel de Containers

Hay múltiples soluciones. Una posible separación es la siguiente

- Front End Web
- Front End Smartphone
- Front End Tablet
- Back End Logistico (órdenes, platos, clientes, repartidores)
- Back End Gestión (cobros a clientes, cobros a restaurants, apoyo a la gestión)
- API Gateway