DP1 2020-2021

Documento de Diseño del Sistema

Proyecto “**Merlantico Vacaciones**”

https://github.com/gii-is-DP1/dp1-2020-g3-14

Miembros:

- CARVAJAL MORENO, PEDRO PABLO
- HIDALGO RODRIGUEZ, JAVIER
- MORENO VAZQUEZ, ENRIQUE
- MUÑOZ CIFUENTES, PEDRO JESUS
- REGADERA MEJIAS, JOSE FRANCISCO
- SEVILLA CABRERA, ALVARO

Tutor: José María García Rodríguez

GRUPO G3-14

Versión 1.0

10/01/2021

Historial de versiones

Estos son ejemplo del contenido que debería tener el historial de cambios del documento a entregar a lo largo de los sprints del proyecto

Fecha	Versión	Descripción de los cambios	Sprint
13/12/2020	V1.1	● Creación del documento	3
13/12/2020	V1.2	● Añadido diagrama de dominio/diseño ● Explicación de la aplicación del patrón caché	3
3/01/2021	V1.2.1	● Modificado diagrama UML	3
3/01/2021	V1.3	● Añadidas las decisiones 1,2 y 3	3
4/01/2021	V1.4	● Añadido diagrama de capas	3
8/01/2021	V1.5	● Añadida Decisión 4 ● Modificados diagramas de capas y UML	3

Contents

Introducción

La principal idea de este proyecto es la de crear una agencia de viajes que se encargue de organizar unas vacaciones completas y no solo se encargue de reservar un vuelo o un desplazamiento concreto, sino que también se puede elegir entre diferentes lugares de hospedaje, complementos dentro del hotel una vez se reserve, y actividades que se realizarán durante las vacaciones, tanto fuera como dentro del propio hotel.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

Todas nuestros modelos heredan de baseEntity por eso hemos omitido las generalizaciones hacia BaseEntity para simplificar el diagrama:

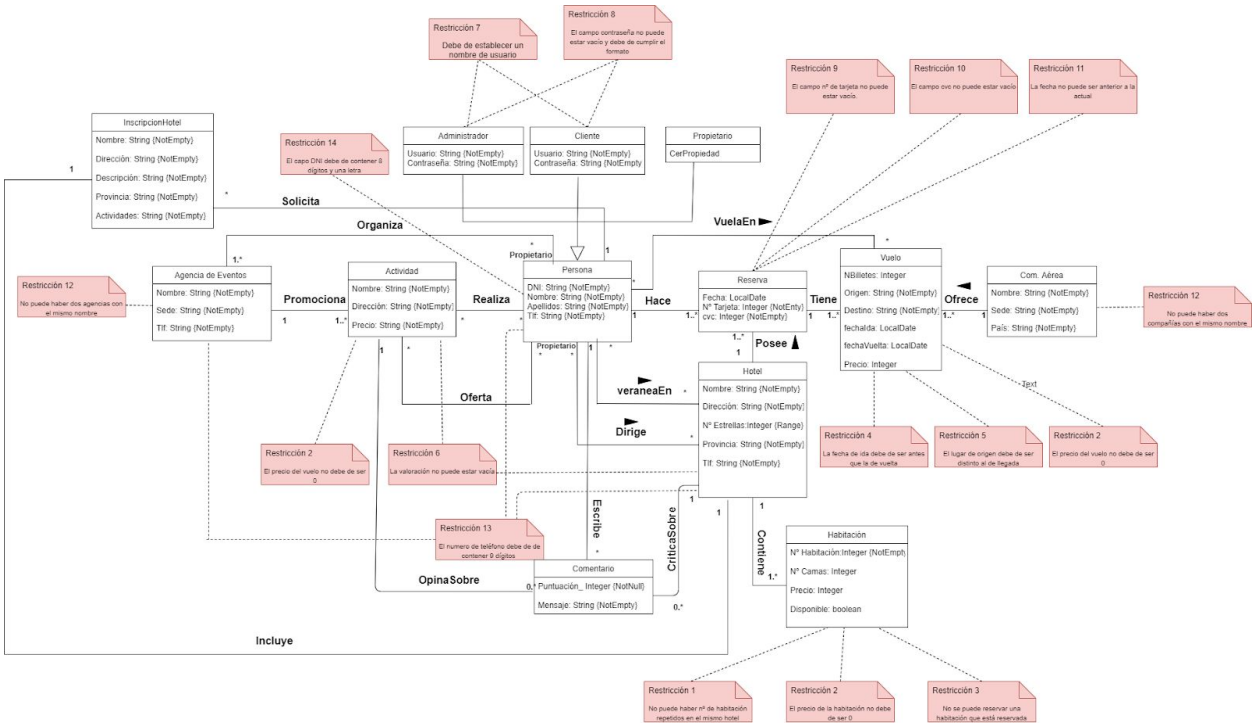
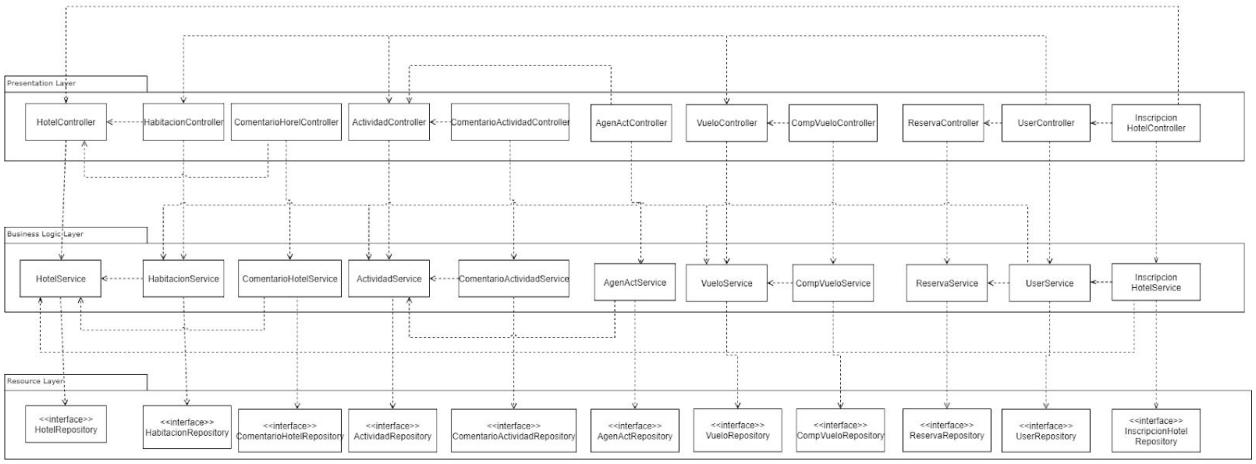


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

A continuación, vamos a mostrar las relaciones entre los controladores, servicios y repositorios divididos en capas que hemos implementado en nuestra aplicación.



Patrones de diseño y arquitectónicos aplicados

Patrón: Modelo Vista Controlador (MVC)

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

Describir las partes de la aplicación donde se ha aplicado el patrón. Si se considera oportuno especificar el paquete donde se han incluido los elementos asociados a la aplicación del patrón.

Todo el proyecto ha sido dividido según el patrón Modelo-Vista-Controlador. Usando los archivos .jsp para las vistas, las clases controller para el controlador y los modelos para el modelo.

Clases o paquetes creados

Indicar las clases o paquetes creados como resultado de la aplicación del patrón.

- model (Modelos): Actividad, AgenAct, Authorities, BaseEntity, ComentarioActividad, ComentarioHotel, CompVuelo, Habitacion, Hotel, Person, ReservaHotel,ReservaVuelo, ReservaActividad, User, Vuelo,InscripcionHotel.
- web(controladores): ActividadController, AgenActController, ComentarioActividadController, ComentarioHotelController, CompVueloController, HabitacionController, HotelController, ReservaController, Search2EntitiesController, UserController, VueloController,ReservaHotelController,ReservaHotelHistorialController, ReservaVueloController,ReservaVueloHistorialController, ReservaActividadController, ReservaActividadHistorialController, InscripcionHotelController.
- service(servicios): ActividadService, AgenActService,AuthoritiesService, ComentarioActividadService, ComentarioHotelService, CompVueloService, HabitacionService, HotelService,InscripcionHotelService, ReservaHotelService, ReservaVueloService, ReservaActividadService, UserService, VueloService.

- repository (Repositorios): ActividadRepository, AgenActRepository, AuthoritiesRepository, ComentarioActividadRepository, ComentarioHotelRepository, CompVueloRepository, HabitacionRepository, HotelRepository, InscripcionHotelRepository, ReservaHotelRepository, ReservaVueloRepository, ReservaActividadRepository, UserRepository, VueloRepository.
- jsp(vistas): Están divididos en carpetas con el nombre de la entidad a la que hacen referencia. Actividades, AgenActs, CompVuelo, Hoteles, inscripcionHoteles, reservasHotel, reservasVuelo, reservasActividad, Search, User, Vuelos.

Ventajas alcanzadas al aplicar el patrón

- Separación clara de dónde tiene que ir cada tipo de lógica, facilitando el mantenimiento y la escalabilidad de nuestra aplicación.
- Sencillez para crear distintas representaciones de los mismos datos.
- Facilidad para la realización de pruebas unitarias de los componentes, así como de aplicar desarrollo guiado por pruebas.
- Reutilización de los componentes.
- Recomendable para el diseño de aplicaciones web compatibles con grandes equipos de desarrolladores y diseñadores web que necesitan gran control sobre el comportamiento de la aplicación.

Decisiones de diseño

En esta sección describiremos las decisiones de diseño que se han tomado a lo largo del desarrollo de la aplicación que vayan más allá de la mera aplicación de patrones de diseño o arquitectónicos.

Decisión 1: Importación de datos reales para demostración

Descripción del problema:

Como grupo nos gustaría poder hacer pruebas con un conjunto de datos reales suficientes, porque resulta más motivador. El problema es que al incluir todos esos datos como parte del script de inicialización de la base de datos, el arranque del sistema para desarrollo y pruebas resulta muy tedioso.

Alternativas de solución evaluadas:

Alternativa 1.a: Incluir los datos en el propio script de inicialización de la BD (data.sql).

Ventajas:

- Simple, no requiere nada más que escribir el SQL que genera los datos.

Inconvenientes:

- Ralentiza todo el trabajo con el sistema para el desarrollo.
- Tenemos que buscar nosotros los datos reales

Alternativa 1.b: Crear un script con los datos adicionales a incluir (extra-data.sql) y un controlador que se encargue de leerlo y lanzar las consultas a petición cuando queramos tener más datos para mostrar.

Ventajas:

- Podemos reutilizar parte de los datos que ya tenemos especificados en (data.sql).

- No afecta al trabajo diario de desarrollo y pruebas de la aplicación

Inconvenientes:

- Puede suponer saltarnos hasta cierto punto la división en capas si no creamos un servicio de carga de datos.
- Tenemos que buscar nosotros los datos reales adicionales

Alternativa 1.c: Crear un controlador que llame a un servicio de importación de datos, que a su vez invoca a un cliente REST de la API de datos oficiales de XXXX para traerse los datos, procesarlos y poder grabarlos desde el servicio de importación.

Ventajas:

- No necesitamos inventarnos ni buscar nosotros los datos.
- Cumple 100% con la división en capas de la aplicación.
- No afecta al trabajo diario de desarrollo y pruebas de la aplicación

Inconvenientes:

- Supone mucho más trabajo.
- Añade cierta complejidad al proyecto

Justificación de la solución adoptada

Como consideramos que la división es capas es fundamental y no queremos renunciar a un trabajo ágil durante el desarrollo de la aplicación, seleccionamos la alternativa de diseño 1.a.

Decisión 2 Mostrar los hoteles por la provincia

Descripción del problema:

Imposibilidad de crear un spinbox o selector para el filtrado de hoteles por la provincia. De forma que si pulsamos o seleccionamos una provincia, la lista de hoteles queda reducida a sólo los hoteles que estuviesen en esa provincia.

Alternativas de solución evaluadas:

Alternativa 2.a: Crear lista por cada provincia en las vistas de forma que un jsp por cada provincia que muestre solo los hoteles de esa provincia.

Ventajas:

- Los hoteles quedarían reducidos a sus provincias

Inconvenientes:

- Supone mucho más trabajo.
- Añade muchísimos archivos casi idénticos, haciendo tedioso el crearlos

Alternativa 2.b: Redirigir la lista de hoteles a otra lista con solo los hoteles que tengan la provincia seleccionada

Ventajas:

- Los hoteles quedarían reducidos a sus provincias

Inconvenientes:

- Habría que desarrollar nuevos métodos en el controlador, repositorio y servicio para obtener los hoteles por la provincia

Justificación de la solución adoptada

Como una historia de usuario nuestra es poder buscar por la provincia la alternativa 2.b porque nos parecía más sencillo implementarlo que hacer el spinbox o selector porque nos estaba dando problemas ya que no podíamos obtener el valor seleccionado para hacer el filtro, de esta forma obtener el valor y reducimos la lista.

Decisión 3 Entidades comentarios

Descripción del problema:

Establecer comentarios y añadirlos en sus respectivas tablas de nuestra aplicación para poder gestionar toda dicha información de forma más rápida y eficiente.

Alternativas de solución evaluadas:

Alternativa 3.a: Crear entidades para los distintos tipo de comentarios que se pueden realizar.

Ventajas:

- Facilidad de mantener y realizar cambios.

Inconvenientes:

- Añade muchísimos archivos casi idénticos, haciendo tedioso el crearlos

Alternativa 3.b: Crear un validador que te redirige al tipo de comentario

Ventajas:

- No tiene archivos idénticos.

Inconvenientes:

- Difícil de desarrollar y mantener.

Justificación de la solución adoptada

Como una historia de usuario hemos decidido que la alternativa 3.a, porque nos parecía más sencillo implementarlo que hacer un validador porque nos estaba dando problemas ya que no podíamos obtener dicha validación.

Decisión 4 Solicitud de inscripción hotel

Descripción del problema:

Realizar una solicitud para que un propietario de hotel pueda enviar los datos de su hotel y que además se muestren las distintas solicitudes realizadas por todos los propietarios.

Alternativas de solución evaluadas:

Alternativa 4.a: Usar la entidad Hotel ya creada y a partir de ahí realizar los formularios de inscripción.

Ventajas:

- No hace falta crear más entidades
- Solo hace falta una vista para el formulario.

Inconvenientes:

- Se muestran todos los hoteles y las inscripciones juntos.
- Se necesitan crear atributos relacionados a las inscripciones dentro de la entidad Hotel para poderlos guardar en la base de datos, lo cual puede ser confuso por los nombres.
- El archivo es más grande, lo que provoca que sea más difícil encontrar los datos y errores.

Alternativa 4.b: Crear una entidad llamada SolicitudInscripcionHotel para realizar un formulario nuevo.

Ventajas:

- Se puede crear el modelo a parte y no se relaciona directamente con los hoteles.
- Es más fácil encontrar errores y datos.

Inconvenientes:

- Datos casi idénticos a los hoteles, lo cual puede resultar un poco confuso.
- Se necesitan crear más vistas para el formulario y la lista de inscripciones.

Justificación de la solución adoptada

Como una historia de usuario hemos decidido optar por la alternativa 4.b, porque nos parece más sencillo de implementar, ya que tenemos una entidad solamente para las inscripciones y se encuentra separada de las demás entidades. A la hora de intentar implementar las inscripciones desde la entidad Hotel nos resultó muy tedioso el mostrar las inscripciones separadas de los hoteles, ya que salían en la misma vista y no conseguimos separarlas.