Entregable # 2 – Arquitectura MVC + Servicios + Docker

	Nombre
Equipo de Trabajo	Abraham Miguel Lora Vargas
	Juan Manuel Young Hoyos

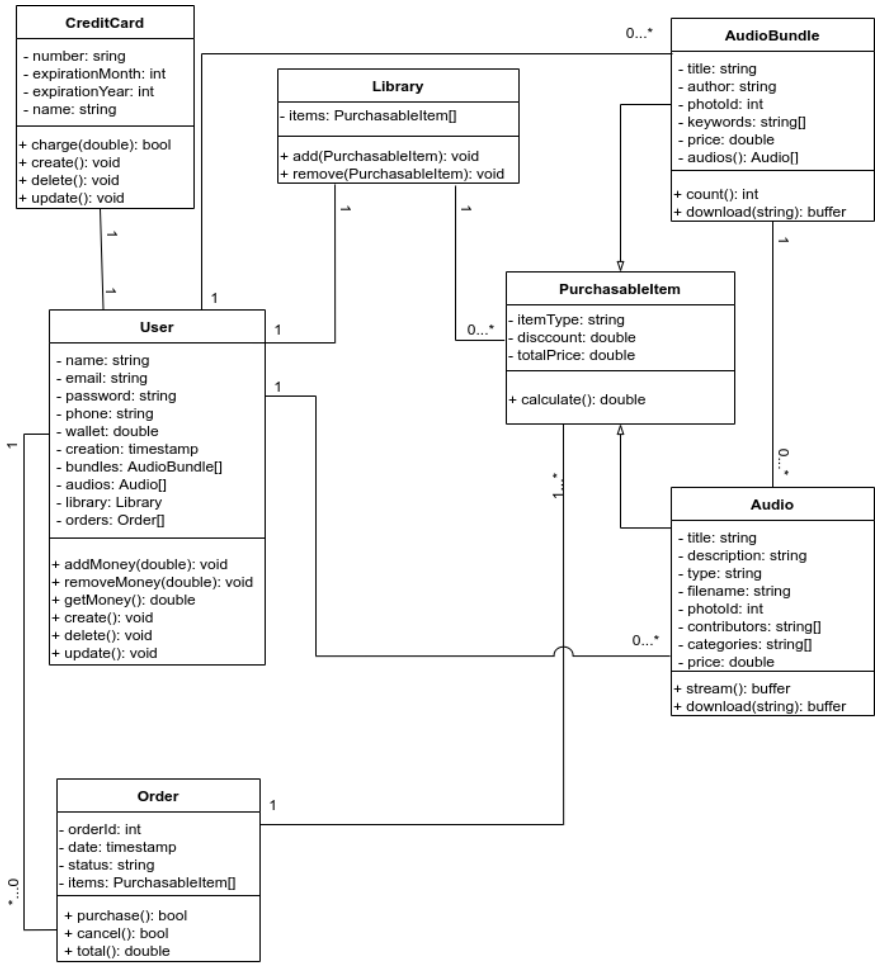
Nombre del proyecto:
_____PlayZ_____

1. Logo equipo

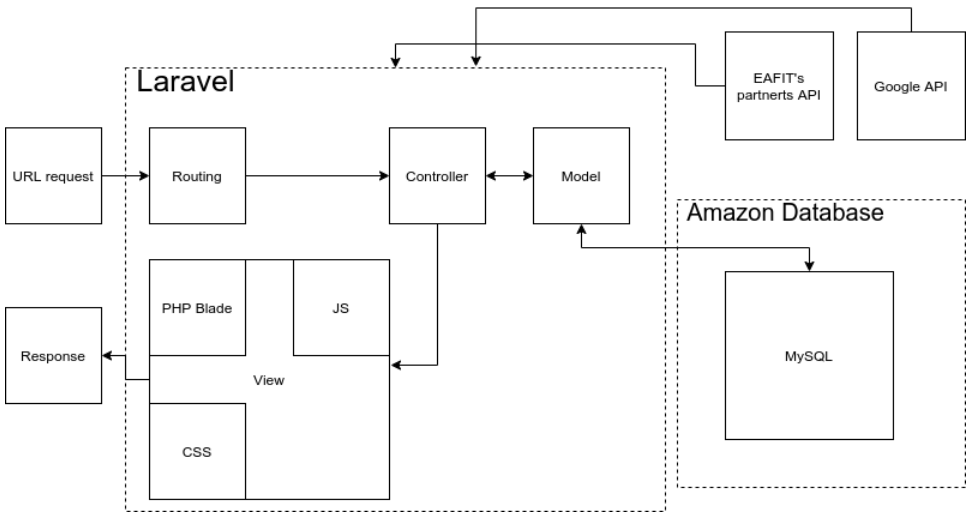
2. Modelo verbal definitivo

PlayZ es un sitio web donde tanto usuarios como creadores de contenidos podrán interactuar con artistas y comercializar sus creaciones, tanto de efectos de sonido, música en general y hasta creaciones MIDI. PlayZ permitirá a todos sus usuarios ser artistas en cualquier momento, basta con crear una publicación de audio, asignarle un precio o dejarla gratuita y ya podrás empezar a comercializar tus creaciones. La plataforma beneficiara tanto a aquellos con talento que quieran vender sus piezas y creaciones musicales como aquellos que estén desarrollando proyectos o sean creadores de contenido, en busca de componentes auditivos para sus creaciones.

3. Diagrama de clases



4. Diagrama de arquitectura



5. Implementación en Laravel

Instrucciones para el arquitecto principal (no puede repetir el arquitecto del proyecto pasado):

- Cree un repositorio en GitHub (rama master).
- Clone el repositorio localmente, luego cree un proyecto en laravel 6.x y súbalo al repositorio GitHub.
- Comparta el repositorio con sus compañeros.
- Repártales a los compañeros las clases que deberán programar. Por ejemplo, si su proyecto consta de 6 clases, repártales de a 2 o 3 clases a cada compañero. Usted como arquitecto tiene total libertad de la repartición de las clases. Recuerde que usted deberá sacar tiempo para la elaboración de unos documentos y para revisar cada código nuevo que intenten aplicar sus compañeros.
- Sus compañeros nunca podrán hacer "push" directo a la rama master.
- Cuando un compañero decida trabajar en el proyecto, deberá: (i) crear una nueva rama, o (ii) realizar un fork del proyecto. Y cuando un compañero desee aplicar los cambios a la rama principal (master o development) deberá realizar un pull request desde GitHub.
- El arquitecto analizará el pull request y solo aceptará los cambios si los considera pertinentes.
- Defina en GitHub un directorio de documentos de codificación. Ese directorio incluirá 2 archivos:
    - **Guía de estilo de programación.** Defina un documento corto (de no más de 2 páginas) con todas las indicaciones de estilo de codificación. Por ejemplo, defina estilo para los ifs, para los while, como nombrar las clases, los métodos, las variables, etc. Ejemplos: https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md - https://guidelines.spatie.be/code-style/laravel-php
    - **Reglas de programación.** Defina otro documento corto (de no más de 2 páginas) con las reglas que usted considera esenciales de programación en su proyecto Laravel. Divida las reglas por categorías (reglas para controladores, para modelos, para vistas, para rutas, etc). Ejemplo de reglas esenciales: (i) nunca haga un echo en un controlador. (ii) Toda ruta debe estar asociada a un controlador. (iii) Toda vista debe extender del layout master. (iv) Todas las vistas deben ser Blade. (v) nunca abra y cierre php dentro de las vistas. Etc.
    - Si un compañero envía un pull request que no siga las reglas definidas en los dos documentos anteriores, remítalo a esa documentación.
- Cree un tablero en asana, trello, o cualquier otro sistema donde pueda controlar las actividades y tareas pendientes del equipo.

Instrucciones para el arquitecto de usabilidad (no puede asignársele esa responsabilidad al arquitecto del proyecto pasado):

- Como arquitecto de usabilidad usted será responsable por que la aplicación sea lo más usable posible. Deberá verificar que todo el sistema desarrollado cumpla con buenos estándares de usabilidad, y deberá asignar a los desarrolladores correspondientes los cambios que considere pertinentes.
- Aspectos para tener en cuenta como arquitecto de usabilidad:
    - Que todas las vistas de la aplicación manejen la misma estructura visual. Que se utilicen los mismos colores, fondos, tipos de letra, etc.
    - Que los formularios estén bien diseñados y contengan una estructura coherente.
    - Que los formularios no se vacíen (y toque volverlos a llenar) si se encuentran errores.
    - Que los campos de los formularios estén bien diseñados (campos de textos, campos de selección, radio buttons, textarea, etc).
    - Que la aplicación tenga botones de fácil acceso (menú principal, menús laterales, pie de página correspondiente, etc).
    - OPCIONAL: Que la aplicación cuente con un sistema de navegación (buscar en Google “breadcumbs navigation”).
    - OPCIONAL: que la aplicación se vea bien, tanto en computador como en celular.
- Cree un tablero en asana, trello, o cualquier otro sistema donde pueda controlar las actividades y tareas pendientes del equipo en cuanto a pendientes de usabilidad.
- Lectura recomendada: https://www.quicksprout.com/website-usability/

Instrucciones globales (entregable #2):

- Implemente todo lo que se pidió para el entregable #1 (si es que aún no lo ha hecho / ver entregable anterior).
- Implemente las 2 funcionalidades interesantes de la entrega anterior (si es que aún no lo ha hecho).
- Haga todas las correcciones que el docente le notificó tanto en las sustentaciones pasadas, como en el documento entregado a cada equipo.
- En esta entrega el sistema se deberá implementar en 2 idiomas (recuerde, nada de textos “quemados” ni en controladores, ni en vistas, siempre use LANG).
- Implemente un servicio web donde usted provea en formato JSON, información relevante de su aplicación.
    - Por ejemplo: (i) lista de productos en stock en venta, incluyendo el enlace directo a la visualización de cada producto; (ii) lista de estudiantes con su nota acumulada; (iii) lista de rutinas de gimnasio, (iv) listado de gafas en promoción, (v) lista de mascotas para adoptar, etc.
    - El equipo siguiente deberá consumir ese servicio, y mostrarlo en la aplicación de ese equipo.
    - Por ejemplo: El equipo 1 provee un servicio donde despliega información de sus productos. El equipo 2 consume ese servicio, crea un nuevo controlador y vista, y crea una ruta: /productos-aliados y ahí

despliega la información que consumió del servicio del Equipo 1. Adicionalmente, el equipo 2 debe proveer un servicio, que deberá ser consumido por el equipo 3, y así sucesivamente.

- Dentro de su aplicación, consuma el servicio del equipo anterior (ver explicación anterior).
- Dentro de su aplicación, consuma un servicio de una compañía tercera (ya sea Google, Facebook, un API en la web, etc). Por ejemplo, el equipo 3, podría consumir un servicio para mostrar el clima actual en Medellín, y desplegarlo en la parte superior de la cabecera de la aplicación.
- Implemente una inversión de dependencias. Sugerencias de inversiones de dependencias:
  - Guardar imágenes de manera local y en S3.
  - Simular pagos con cheques (mostrar un PDF con la info del cheque para pagar), o pagos donde le reste dinero a la cuenta del usuario (simulando que el usuario tiene un atributo con su dinero disponible).
  - Generar reportes en PDF y Excel.
  - Etc.
- Despliegue la aplicación con Docker (en AWS). Mas adelante hay un taller no calificable donde se explica cómo hacerlo.
- Implemente un sistema de nombre de dominio simplificado (que no toque acceder por la IP de Amazon) -> utilice dominios gratis .tk http://www.dot.tk/.
- Implemente mejoras que considere pertinentes, paginación, carga de archivos, mejora en los sistemas de fakers, menús laterales, un sistema de banner en la página principal, entre otros.

Instrucciones de entrega para el arquitecto principal:

- Suba el proyecto a la cuenta que se le brindó en AWS (despliéguelo con Docker).
- Cree un .zip de todo el proyecto y súbalo por Teams (en caso de que no se pueda por cuestiones de tamaño, envíelo por wetransfer o mega al correo del docente).
- Suba este documento al repositorio de GitHub. Y finalmente comparta tanto el link del repositorio, como el dominio .tk con el docente antes de la fecha de finalización de entrega.