Proyecto I

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Redes (IC 4302)
Primer Semestre 2023

1. **Objetivo General**
   - ➔ Desarrollar un prototipo de aplicación que utilice bases de datos SQL y NoSQL en Azure Cloud y Google Cloud Platform.
2. **Objetivos Específicos**
   - ➔ Desarrollar una aplicación para dispositivos móviles mediante la herramienta Thunkable.
   - ➔ Implementar una base de datos en tiempo real mediante la herramienta Firebase.
   - ➔ Implementar una base de datos SQL mediante Azure SQL.
   - ➔ Implementar una base de datos NoSQL mediante Azure Cosmos.
   - ➔ Automatizar una solución mediante el uso de Docker.
   - ➔ Implementar un manejador de archivos en Node.js y Azure Blob Storage.
   - ➔ Desarrollar un API en el lenguaje de programación Python.
3. **Datos Generales**
   - ➔ El valor del proyecto: 25%
   - ➔ Nombre del proyecto: EduHub.
   - ➔ La tarea debe ser implementada en grupos de máximo 5 personas.
   - ➔ La **fecha de entrega** es 21/04/2023 antes de las 6:00 pm.
   - ➔ Cualquier indicio de copia será calificado con una nota de 0 y será procesado de acuerdo con el reglamento. La copia incluye código/configuraciones que se puede encontrar en Internet y que sea utilizado parcial o totalmente sin el debido reconocimiento al autor.
   - ➔ La revisión es realizada de forma virtual mediante citas en las cuáles todos los y las integrantes del grupo deben estar presentes, el proyecto debe estar completamente automatizado, el profesor decide en que computadora probar.
   - ➔ Se debe incluir documentación con instrucciones claras para ejecutar el proyecto.
   - ➔ Se deben incluir pruebas unitarias para verificar los componentes individuales de su proyecto, si estas no están presentes se obtendrá una nota de 0.
   - ➔ Se espera que todos y todas las integrantes del grupo entiendan la implementación suministrada.
   - ➔ Se deben seguir buenas prácticas de programación en el caso de que apliquen, entre ellas:
     - ◆ Documentación interna y externa.
     - ◆ Estándares de código.
     - ◆ Diagramas de arquitectura.
     - ◆ Diagramas de flujo
     - ◆ Pruebas unitarias.
   - ➔ Toda documentación debe ser implementada en Markdown.

- ➔ Si el proyecto no se encuentra automatizado obtendrá una nota de 0. Si el proyecto no se entrega completo, la porción que sea entregada debe estar completamente automatizada, de lo contrario se obtendrá una nota de 0.
- ➔ En caso de no entregar documentación se obtendrá una nota de 0.
- ➔ El email de entrega debe contener una copia del proyecto en formato tar.gz y un enlace al repositorio donde se encuentra almacenado, debe seguir los lineamientos en el programa de curso.
- ➔ Los grupos de trabajo se deben autogestionar, la persona facilitadora del curso no es responsable si un miembro del equipo no realiza su trabajo.
- ➔ En consulta o en mensajes de correo electrónico o Telegram, como mínimo se debe haber leído del tema y haber tratado de entender sobre el mismo, las consultas deben ser concretas y se debe mostrar que se intentó resolver el problema en cuestión.

4. **Descripción**

EduHub es una primera versión de aplicación que tendrá las siguientes características (**no todas serán implementadas al 100%):**

- Roles: Los roles en la aplicación definirán los permisos que tendrá cada usuario, en esta versión contaremos con los siguientes roles:
  - admin: tienen acceso a toda la aplicación, pueden cambiar cualquier aspecto de esta.
  - profesor: tiene permisos para agregar cursos y modificar los mismos, incluye todos los permisos del rol estudiante.
  - asistente: es un estudiante que puede ser asignado a un curso como asistente, el mismo tendrá los mismos permisos que un profesor.
  - estudiante: puede realizar el proceso de matrícula y subir soluciones de actividades.
- Usuarios: Cada usuario se registra en la aplicación únicamente mediante Thunkable, la información del usuario tiene que estar almacenada en Firebase, cada grupo deberá definir y documentar que información se va a recolectar para cada usuario, por defecto existirá un usuario administrador llamado ***root***, la contraseña la define cada grupo, es importante mencionar que solo un administrador puede nombrar otros administradores, en el momento del registro, un usuario selecciona si es estudiante o profesor. Solo un profesor puede nombrar un asistente.
- Administración de archivos: Un usuario registrado mediante Thunkable puede ingresar a la aplicación Files Manager para administrar archivos:
  - Crear: Se les genera un identificador único.
  - Borrar: Un archivo no se puede borrar si es referenciado por otra parte de la aplicación.
  - Modificar.
  - Descargar.

  Para cada archivo se debe guardar, el usuario al cual pertenece el archivo, tipo, periodo, curso fecha de creación, fecha de última modificación, nombre y descripción, es importante

mencionar que se debe implementar versionado de cada archivo, la versión ***latest***, siempre apunta a la última versión de este.

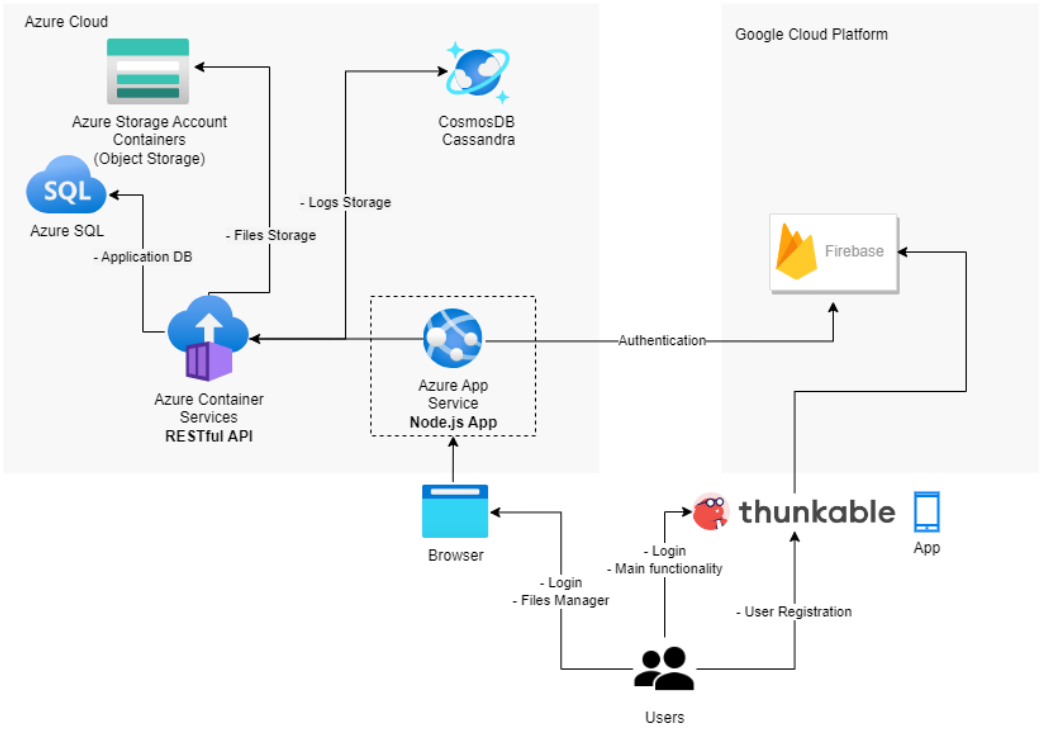
- Administración de Periodos Lectivos: Un administrador podrá crear/borrar/modificar periodos, los mismos tendrán las siguientes características:
    - Id: Se genera automáticamente.
    - Tipo: anual, semestral, trimestral, cuatrimestral o bimestral.
    - Inicio: fecha de inicio.
    - Final: fecha de finalización.
    - Estado: Podría ser: creado, en progreso o finalizado. Si se encuentra finalizado, no se podrá modificar ningún tipo de información asociado con este.
- Administración de Escuelas: Un administrador podrá crear/borrar/modificar escuelas, la información requerida de esta será definida por cada grupo.
- Administración de Cursos: Un administrador podrá crear/borrar/modificar cursos, estos tienen un código, un nombre, tipo (los mismos tipos de los periodos), una cantidad de créditos, escuela asociada, cantidad de horas de clases por semana, una descripción y archivos asociados.
- Administración de carreras: Un administrador podrá crear/borrar/modificar carreras, las mismas tienen un nombre, una escuela asociada, una descripción y uno o varios documentos asociados.
- Administración de planes: Un administrador puede crear/borrar/modificar planes, los mismos tienen un estado (creación, activo, cerrado), numero de plan, carrera asociada, fecha creación, fecha en que fue activado, fecha de finalización, es posible asociar cursos con un plan y entre estos cursos se pueden definir requisitos o dependencias.
- Administración de grupos: Un administrador puede crear/borrar/modificar grupos, estos se asocian con un curso, periodo (los tipos de los periodos del curso y el periodo con el cual se está asociando, deben ser iguales) y profesor (se verifica que el profesor y el curso pertenezca a la misma escuela), se genera un identificador automáticamente, además se le pueden agregar horarios (no puede superar las cantidades de horas por semana definidas en el curso), puede asociar documentos y se le indica un número máximo de estudiantes. Un profesor puede hacer las mismas acciones que un administrador, pero únicamente podrá crear cursos para sí mismo, además podrá abrir el curso y cerrar el curso.
- Un administrador puede asociar estudiantes con planes.
- Un administrador puede asociar profesores con escuelas.
- Administración de periodos de matrícula: Un administrador podrá crear/borrar/modificar periodos de matrícula, un periodo de matricula se puede abrir y se puede cerrar, un periodo de matricula se encuentra asociado con únicamente un periodo lectivo, el mismo define un “nombre amigable”. En el momento que se abre un periodo de matrícula, los estudiantes podrán matricularse en los cursos que se encuentran asociados al periodo lectivo, en el momento que se cierra el periodo de matricula los estudiantes no podrán hacerlo.

- Administración de evaluación: Un profesor en sus grupos podrá agregar/modificar/borrar la evaluación del grupo, en este caso agregará rubros y el porcentaje asignado a este, la suma de los porcentajes no podrá ser mayor a 100%.
- Administración de actividades: Un profesor en sus grupos podrá agregar actividades, las mismas se asocian con un rubro de evaluación, cada actividad tendrá un porcentaje asociado, una fecha de entrega y la suma de todas las actividades asociadas con un rubro no podrán superar el porcentaje asociado al rubro de evaluación.
- Evaluación: Se tendrán los siguientes puntos de vista:
    - Estudiante: Para cada una de las evaluaciones los estudiantes deberán subir un archivo y en el sistema tendrá que asociar el archivo con una actividad, este debe ser asociado antes de la fecha de entrega de la evaluación, de lo contrario se asume nota de 0.
    - Profesor: Podrá realizar una evaluación luego de que ha expirado la fecha de entrega, podrá seleccionar algún estudiante y evaluar una actividad, en la evaluación se puede agregar comentarios, asociar archivos y dar una nota entre 0 y 100, esta automáticamente calculará el porcentaje obtenido por la evaluación. Un profesor podrá modificar cualquier nota siempre y cuando el periodo lectivo no se encuentre cerrado.

  Un estudiante podrá ver sus evaluaciones y resultados en todo momento, así como la nota parcial del curso, el profesor podrá hacer lo mismo con cualquier estudiante en el grupo.
- Evaluación del curso/grupo: Se podrá realizar la evaluación del curso/grupo, los parámetros y diseño de la evaluación serán definidos por cada grupo, es completamente anónimo.
- Un administrador puede realizar el cierre de un periodo lectivo, en este caso el sistema calcula las notas finales de todos los estudiantes matriculados en un curso, así como su hora de matrícula del próximo periodo.
- Horario de matrícula: Un estudiante al registrarse en el sistema tendrá un horario de matricula de 100, esto significa que podrá matricular desde el momento que se abre el periodo de matricula (si se abre un lunes a las 7:00 am, podrá matricular a esta hora), conforme pasan los periodos lectivos, el horario de matrícula estará afectado por el promedio de notas del último periodo lectivo cursado (sin importar el tipo), esto quiere decir que si el promedio fue de un 70 su horario de matrícula será este, a nivel de sistema esto se interpreta de la siguiente forma, todos los estudiantes con hora de matricula entre 95 y 100, podrán matricular desde el momento que se abre la matricula, quienes tengan una hora de matricula entre 95 y 90, podrán matricular desde el momento que se abre la matricula mas una hora (si se abre a las 7:00 am, este grupo podrá matricular a partir de las 8:00 am), esta dinámica se mantendrá igual hasta que se llega al horario de matricula igual a 70 en ese momento cualquier estudiante podrá matricular.
- Todo el sistema guardara en un log toda la actividad realizada por los usuarios.
- Matricula: Desde el momento que se abre un proceso de matrícula, si su horario de matrícula se lo permite y además ha aprobado todos los requisitos (plan asociado), un estudiante se podrá matricular o des matricular de un grupo, el sistema deberá validar que no existe un choque de horarios y que no se encuentra matriculando en el mismo curso en diferentes grupos.

- Si no existe un campo en un grupo, el estudiante podrá realizar una matricula tentativa, consiste en que deberá seleccionar otro grupo del mismo curso y marcar el grupo en el que desea campo como tentativo (el sistema valida que no existe choque). Si en algún momento se liberan mas campos (incrementa el cupo del grupo) o alguna persona cambia su matricula y libera un campo, el sistema automáticamente realiza la matricula tentativa, esta consiste en que libera el grupo en el que se había matriculado tentativamente y matricula en el grupo que marcó como tentativo, se comporta de forma FIFO, el primer estudiante en marcar como tentativa una matricula es quien recibe el campo.
- Un estudiante puede estar en varios planes de diferentes carreras activo.
- Se podrá visualizar toda la información del curso/grupo cuando se este realizando la matricula incluso la evaluación de iteraciones pasadas del curso que fueran impartidas por el mismo profesor.
- El sistema permitirá generar reportes de matricula los cuales pueden ser solicitados por el profesor o administrador.
- Un estudiante únicamente podrá ver los cursos que puede matricular.

Para efectos del prototipo de la primera versión que será implementada en este proyecto, se utilizará la siguiente arquitectura:



Los requerimientos de este prototipo son:

- Crear un diseño/diagrama entidad relación de la primera versión de la aplicación EduHub (descrita anteriormente), se insta a cada grupo a realizar cualquier ajuste a la primera versión de EduHub basado en su experiencia con sistemas similares.
    - Para este diagrama se deben seguir buenas practicas en el diseño de bases de datos, entre estas se encuentran normalización y escoger adecuadamente los tipos de datos.
    - Se debe contar con un estándar para nombrar los componentes.
    - Como entregable se espera un diagrama con su respectiva explicación.
        - Si en la descripción anterior un grupo considera que se ocupa más información para poder completar el diagrama, se puede agregar una sección de suposiciones, por ejemplo, en la sección que dice "… se genera un identificador automáticamente, además se le pueden agregar horarios (no puede superar las cantidades de horas por semana definidas en el curso) …", no se especifica como se representaran los horarios en este caso cada grupo asume como será representado.
- Crear una base de datos en Azure SQL que refleje el diagrama entidad relación creado en el paso anterior.
    - Como entregable se espera un archivo .sql que describa la base de datos.
- Generar datos de prueba para llenar la base de datos.
    - Como entregable se espera un archivo .sql con los statements para cargar datos en la base de datos.
- Implementar el registro de usuarios y Login mediante Thunkable y Firebase:
    - La información que se solicitará al usuario la define cada grupo y estará almacenada únicamente en Firebase.
    - En este paso, Firebase genera un id de usuario único, este será usado en otras bases de datos para asociar la información con un usuario especifico.
    - Como entregable se espera una copia del proyecto de Thunkable.
- Implementar una aplicación en Node.js que se ejecute en Azure Container Services, la misma permitirá realizar la administración de archivos y se autenticara mediante Firebase, toda la información de los archivos se encuentra almacenada en la base de datos de Azure SQL y los archivos se encuentran almacenados en Azure Storage Account Containers. Si algún grupo desea ejecutar esta aplicación en Azure App Service se le reconocerá un 3% extra sobre el porcentaje de este proyecto por lograrlo.
- Implementar el módulo de sistema de matricula mediante Thunkable, el mismo consumirá Web Services del REST API, la funcionalidad de matricula fue descrita en la sección anterior.
    - Como entregable se espera una copia del proyecto de Thunkable, puede ser el mismo proyecto en el cual se implementa el registro y login de usuario.
- Implementar un REST API en Python el cual debe estar corriendo en Azure Container Services, este implementará cualquier funcionalidad requerida por la aplicación de Thunkable o la aplicación en Node.js. Se debe usar consistentemente los verbos HTTP para diferentes acciones (PUT=Create, POST=modificar, GET=Obtener y DELETE=Borrar), este API recibe como parámetro

el UserId (obtenido de Firebase), este permitirá verificar que se trata de un usuario valido, verificar si se tienen permisos y asociar cualquier tipo de información.

- Toda acción realizada por un usuario en la aplicación deberá ser almacenada Azure Cosmos DB, como un documento JSON, esto permitirá realizar auditoria.

Infraestructura en Azure

Es importante mencionar que los grupos no deberán crear la infraestructura en Azure de forma manual, el profesor proporcionará scripts de infraestructura como código para generar la infraestructura requerida, cada grupo deberá modificar algunos parámetros para generar su propia infraestructura y apuntar a sus imágenes de Docker, para estos efectos deberán instalar las herramientas Terraform y Azure CLI. El profesor brindará una demostración de cómo crear la infraestructura.

Azure App Service

Si algún grupo desea utilizar Azure App Service para correr su aplicación NodeJS y se logra satisfactoriamente, se reconocerá un 3% extra sobre el valor de este proyecto, para esto se debe integrar el componente de Terraform azurerm_linux_web_app al código de la infraestructura, en esta pagina pueden encontrar un ejemplo de cómo hacerlo.

Documentación

Se deberá entregar una documentación que al menos debe incluir:

- Instrucciones claras de como ejecutar su proyecto.
- Pruebas realizadas, con pasos para reproducirlas.
- Resultados de las pruebas unitarias.
- Recomendaciones y conclusiones (al menos 10 de cada una).
- La documentación debe cubrir todos los componentes implementados o instalados/configurados, en caso de que algún componente no se encuentre implementado, no se podrá documentar y tendrá un impacto en la completitud de la documentación.
- Referencias bibliográficas donde aplique.

Imaginen que la documentación está siendo creada para una persona con conocimientos básicos en el área de computación y ustedes esperan que esta persona pueda ejecutar su proyecto y probarlo sin ningún problema, el uso de imágenes, diagramas, code snippets, videos, etc. son recursos que pueden ser útiles.
La documentación debe estar implementada en Markdown y compilada en un PDF.

5. **Recomendaciones**
   - Utilizar telnet/curl/postman para interactuar con los componentes y verificar que los mismos están funcionando correctamente.

- Utilizar Docker Desktop para instalar Docker en Windows.
- Realicen peer-review/checkpoints semanales y no esperen hasta el último momento para integrar su aplicación.
- Crear un repositorio de GitHub e invitar al profesor.
- Realizar commits y push regulares.
- Para la aplicación de Node.js se recomienda usar algún tipo de JS Framework como VueJS. Existen cantidades de tutoriales de cómo hacer autenticación con Firebase, así como un File Manager.
- Para la revisión, se recomienda tener clientes de administración de las bases de datos para poder realizar consultas y observar su comportamiento.

6. **Entregables**

- Documentación.
- Documentación con las pruebas unitarias del API.
- Diagrama entidad/relación junto con su explicación.
- Archivos .sql para crear la base de datos y generar datos de prueba.
- Docker files y código para generar el RESFUL API y la aplicación Node.JS
- Archivos de infraestructura como código con los cambios requeridos por cada grupo (en especial si se usa Azure App Service).
- Enlace del proyecto de Thunkable.

7. **Evaluación**

Funcionalidad / Requerimiento	Porcentaje
Documentación (*)	10%
Pruebas unitarias (*)	10%
Diagrama entidad relación	20%
Archivos .SQL de base de datos y datos prueba.	10%
RESTFUL API	20%
Node.js App	15%
Thunkable App	15%
	100

(*) La completitud de la documentación, así como pruebas unitarias se encuentra acotado por la completitud de otros requerimientos.