Prueba de Concepto (TP0) - Lector de Notas

Motivación 1

Contexto 1

Objetivo 2

Anexo 1: Introducción a la arquitectura y tecnologías involucradas 3

Anexo 2: ejemplos de consultas 6

Cargar tus datos de estudiante 6

Consultar tus datos de estudiante 6

Consultar tus asignaciones y notas 6

Anexo 3: Tecnologías útiles 8

Motivación

La motivación de este Trabajo Práctico "Prueba de Concepto" es dar un vistazo rápido sobre varias de las tecnologías e ideas arquitectónicas que se emplearán durante el año:

- Integración con un sistema externo a través de HTTP
- Patrón MVC (Model View Controller)
- Java y su ecosistema de desarrollo (maven, Eclipse, Git)

Además de construir una aplicación que, aunque muy simple, será completamente funcional y tendrá una utilidad.

Contexto

Para los docentes y estudiantes, la gestión de las notas es una tarea importantísima: los docentes tienen que calificar las distintas tareas de los estudiantes, y estos a su vez, están ciertamente interesados en saber qué nota obtuvieron.

Entonces, ¿cómo podemos cargarlas y mostrarlas a nuestros estudiantes? Tradicionalmente, mediante una planilla en papel y comunicando las notas personal, pero con los tiempos que corren, se ha vuelto muy frecuente informatizar esta tarea, utilizando planillas on line, como por ejemplos las de Google Docs.

¿Y qué problema hay con ellas? Primero, para los docentes es fácil equivocarse: ingresando notas con un formato incorrecto (porque las hay numéricas y conceptuales), haciéndolo para una tarea equivocada, intercambiando estudiantes, etc. Y segundo, por motivos de privacidad de datos, una nota debería ser solo notificada a quien la obtuvo y no estar pública, tal como en los viejos tiempos del papel y el lápiz.

Por eso es que los docentes de Diseño de Sistemas no tuvieron mejor idea que construir un sistema web para cargarlas, con el siguiente modelo de dominio:

- Hay muchos estudiantes, de los que se sabe su nombre y legajo
- Hay muchas tareas, que el docente puede definir, como parciales y trabajos prácticos. El docente establece un tipo de calificación para estas tareas: numéricas o conceptuales (letras como M, B- y R+)
- Cada estudiante tiene muchas asignaciones de tareas, que son las relaciones entre una tarea y el estudiante, y para cada asignación, puede tener muchas notas, una por cada vez que se evaluó (por ejemplo, en un parcial podes tener hasta 3 notas)
- No se maneja el concepto de curso (al menos por ahora)

El sistema ya está desplegado en notitas.herokuapp.com y parece funcionar correctamente: es una aplicación web a la que sólo ellos tienen acceso. Pero, quizás por descuido, maldad o simplemente didáctica, los docentes no se preocuparon por hacer pantallas para que los estudiantes accedan. En lugar de eso expusieron una interfaz HTTP[1], que expone las siguientes rutas:

- PUT /student: permite actualizar los datos del estudiante autenticado
- GET /student: permite obtener los datos del estudiante autenticado
- GET /student/assignments: permite obtener las asignaciones del estudiante, y cada una de sus notas

Y le darán a cada estudiante un token, único e intransferible, con el que podrán acceder al mismo, para consultar sus notas en cada asignación y actualizar sus datos de perfil[2].

Objetivo

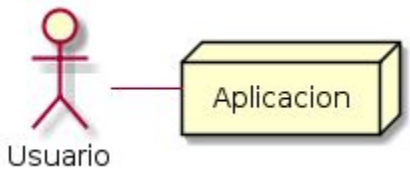
El objetivo principal es una aplicación de escritorio desarrollada en la tecnología Java 8 que permita, para un estudiante:

- Autenticarse mediante un token
- Ver su nota actual para cada asignación, y saber si la aprobó.
- Bonus: actualizar sus datos de estudiante: nombre y apellido, legajo y usuario github.

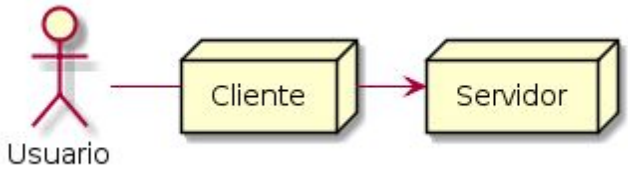
[1] Más adelante veremos que se trata en realidad de una interfaz REST
[2] Consultá con tu docente en clase sobre cómo obtenerlo

Anexo 1: Introducción a la arquitectura y tecnologías involucradas

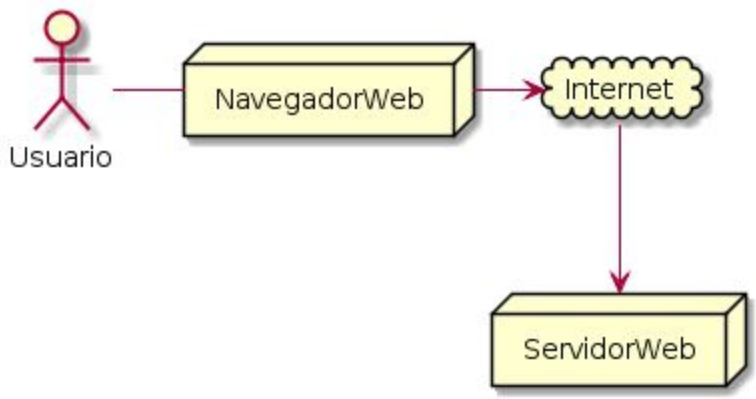
Hasta ahora, en materias como Algoritmos y Estructuras de Datos, y Paradigmas de Programación, siempre construimos aplicaciones con una arquitectura muy simple: un único programa (ya sea una única máquina virtual, intérprete o ejecutable nativo) que resolvía todos los problemas de nuestro sistema. Es decir, su arquitectura (conocida como *arquitectura monolítica*) era trivial:



Sin embargo, muchas aplicaciones son más complejas, y están conformadas por más de un programa, que se ejecuta en más de una computadora[3]. Este tipo de arquitecturas son conocidas como *arquitecturas distribuidas*, y la más simple de ellas es la *cliente-servidor*, en la que hay, al menos, dos programas conectados a través de una red:

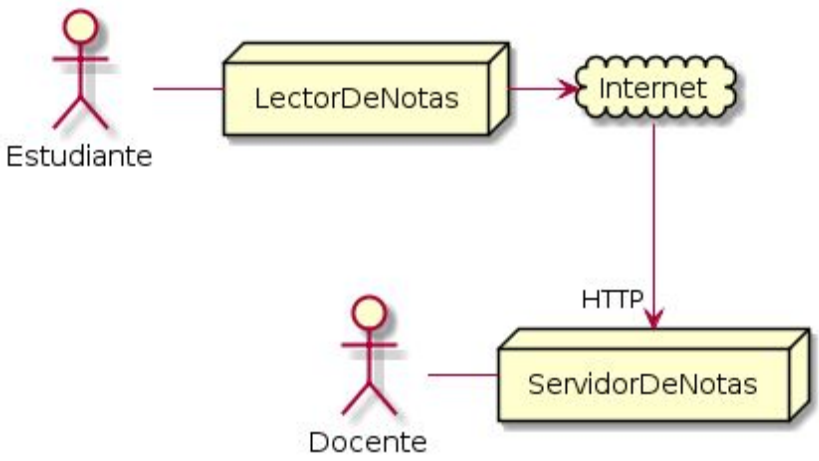


En este tipo de aplicaciones, el programa cliente debe de alguna forma comunicarse con el programa servidor, utilizando algún *protocolo de comunicación*[4]. Por ejemplo, si el protocolo de comunicación es HTTP, y el cliente es un navegador o *browser* (como Google Chrome o Mozilla Firefox), tenemos los fundamentos de la arquitectura de la *Web:*



El sistema que proponemos en este trabajo práctico presenta una arquitectura similar:

[3] ¿Por qué hacer esto? Distribuir la aplicación presenta algunas ventajas, que veremos a lo largo del año.
[4] Más sobre esto, a mediados de año y en las materias Comunicaciones y Redes



Como se puede apreciar, existirá un Servidor de Notas (ya desarrollado, disponible en http://notitas.herokuapp.com), y un Lector de Notas (o mas bien, varios, uno por cada estudiante), que deberemos desarrollar en equipos. La tecnología con la que el lector se comunicará al servidor será HTTP.

¿Y qué es el protocolo HTTP? Una forma de comunicarse a un servidor mediante operaciones muy simples, todas ellas basadas en texto (es decir, no tendremos que manipular bytes, sino strings). Para este trabajo práctico necesitaremos conocer dos:

- el método GET, que nos permite obtener información de un *recurso.* En este caso, contamos con dos recursos:
    - /student: el perfil del estudiante
    - /student/assignments: las asignaciones del estudiantes
- el método PUT, que nos permite actualizar la información de un recurso. En este caso, podemos actualizar el recurso /student.

En HTTP toda la comunicación está estructurada en términos de pedidos (*request*) y respuestas (*response*): el cliente hace un pedido al servidor, y éste le responde (¡nunca al revés!).

Como se verá en el siguiente anexo, los recursos están modelados como *diccionarios de datos* utilizando un formato muy simple y popular hoy en día: JSON. Por ejemplo, el siguiente diccionario representa a una estudiante de nombre María Pérez, usuario de Github mperez y legajo 1421669.

```
{
  "code": "1421669",
  "first_name": "Maria",
  "last_name": "Pérez",
  "github_user": "mperez"
}
```

Como última salvedad a tener en cuenta, una forma bastante común de autenticarse (es decir, indicar al sistema quién está comunicando) en HTTP es mediante el *header Authorization*. Un header es una forma de pasar información complementaria en la comunicación, y Authorization en particular sirve para pasar información para autenticarse, como por ejemplo el token que los docentes proveerán.

Anexo 2: ejemplos de consultas

Cargar tus datos de estudiante

```
curl -i -X PUT http://notitas.herokuapp.com/student \
      -H 'Authorization: Bearer ACA_VA_TU_TOKEN_PERSONAL'
       --data'{"code":"1214731",
               "first_name":"Franco",
               "last_name":"Bulgarelli",
               "github_user":"flbulgarelli"}'

HTTP/1.1 201 Created
```

Consultar tus datos de estudiante

```
curl -i -X GET http://notitas.herokuapp.com/student \
       -H 'Authorization: Bearer ACA_VA_TU_TOKEN_PERSONAL'

{
  "code": "1214731",
  "first_name": "Franco",
  "last_name": "Bulgarelli",
  "github_user": "flbulgarelli"
}
```

Consultar tus asignaciones y notas

```
curl -i -X GET http://notitas.herokuapp.com/student/assignments \
       -H 'Authorization: Bearer ACA_VA_TU_TOKEN_PERSONAL'

{
  "assignments": [
    {
      "id": 1,
      "title": "Primer Parcial",
      "description": null,
      "grades": [
        {
          "id": 1,
          "value": 2,
          "created_at": "2017-03-25T13:56:07.526Z",
          "updated_at": "2017-03-25T13:56:07.526Z"
        },
        {
          "id": 2,
          "value": 7,
          "created_at": "2017-03-25T13:56:07.595Z",
          "updated_at": "2017-03-25T13:56:07.595Z"
        }
      ]
    },
    {
      "id": 3,
      "title": "TPA1",
```

```
      "description": "Primera Entrega del TP Anual",
      "grades": [
        {
          "id": 4,
          "value": "B+",
          "created_at": "2017-03-25T13:56:07.649Z",
          "updated_at": "2017-03-25T13:56:07.649Z"
        }
      ]
    }
  ]
}
```

Anexo 3: Tecnologías útiles

Si bien el protocolo HTTP (Hypertext Transfer Protocolo) es ciertamente complejo (ver por ejemplo la especificación 1.1, que incluye sólo algunos de sus aspectos modernos), afortunadamente no es necesario conocerlo en detalle para realizar algunos primeros programas, dado que existen cientos de bibliotecas y frameworks que nos abstraen de sus complejidad. Por ejemplo, utilizando la biblioteca *Jersey*, acceder al recurso /student mediante el método GET es fácil:

```
Client.create()
      .resource("http://notitas.herokuapp.com")
      .path("student")
      .header("Authorization", "Bearer[5] ACA_VA_TU_TOKEN")
      .accept(MediaType.APPLICATION_JSON)
      .get(ClientResponse.class);
```

Más información:

- Ejemplos de uso de Jersey: Jersey es una tecnología que permite, desde Java, consumir una interfaz HTTP.
- Documentación de Arena: Arena es un framework para construir interfaces gráficas de escritorio en Java. Es muy limitado, pero también muy simple y fácil de entender, suficiente para realizar aplicaciones simples como ésta.
- cURL / Postman: Son herramientas que permiten acceder a una interfaz HTTP sin tener que construir un programa. El primero está basado en texto, el segundo viene con una interfaz gráfica.

[5] La palabra "Bearer" es un detalle del protocolo que no nos importa demasiado por ahora; tan sólo nos limitaremos a colocar esa palabra dentro del header Authorization, delante de nuestro token.