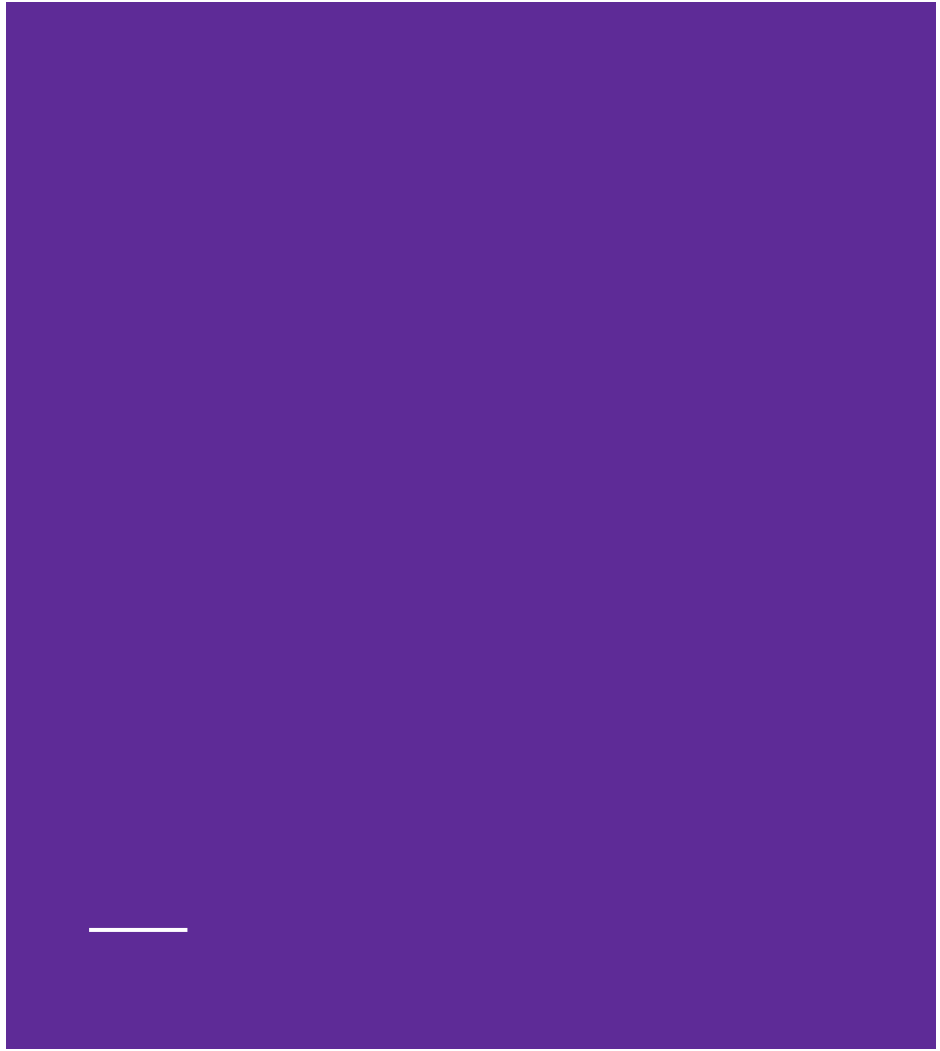
Programación Avanzada
IIC2233 2025-2

Cristian Ruz - Pablo Araneda - Francisca Ibarra - Tamara Vidal - Daniela Concha

Experiencia 4

Serialización y Networking 1
(JSON y Webservices)



Experiencia: ¿Qué vamos a hacer?

1. Aplicaremos el patrón de diseño: ***front-end*/*back-end***.
2. Crearemos una aplicación de múltiples componentes que interactúan entre ellos mediante el **uso de señales**.
3. Posicionaremos elemento en una ventana a través de:
   a. **Coordenadas**
   b. ***Layouts***

DCCitas

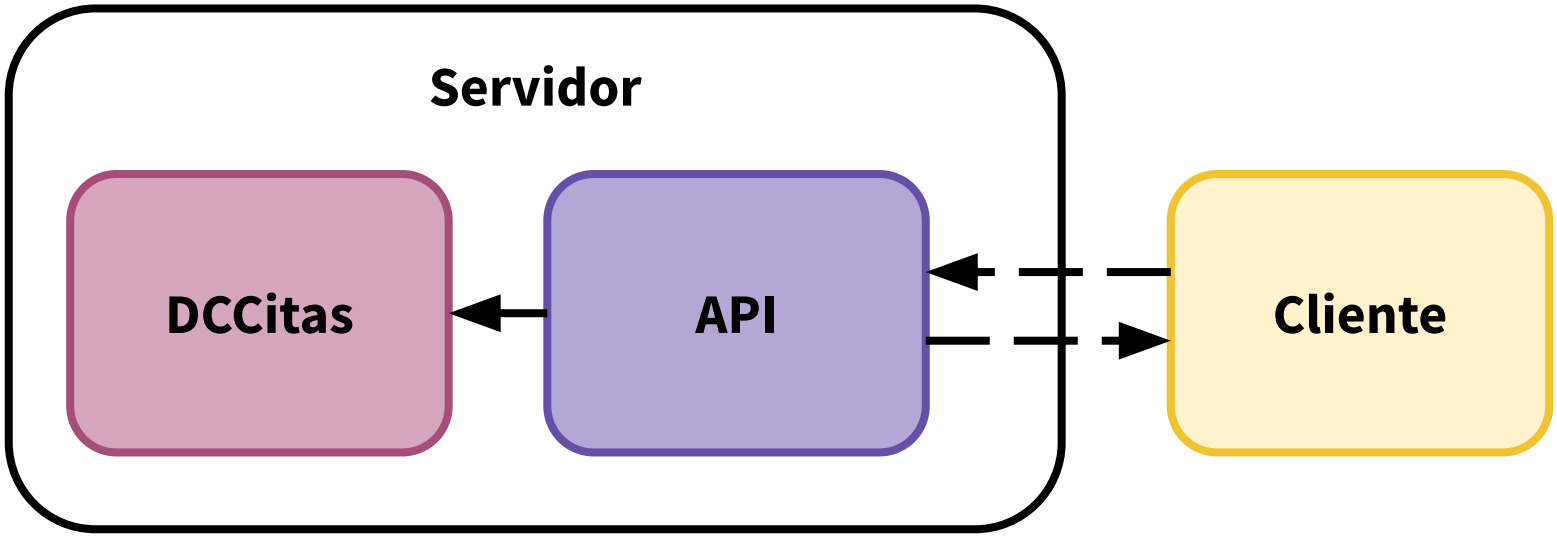
Cansado de una vida de soltería (llevas 2 días sin pareja), decides programar una **aplicación de citas**.

La aplicación utiliza un sistema revolucionario donde se le asignan valores (a.k.a. coordenadas) a cada usuario y a partir de ellas encuentra a los candidatos más apropiados.

¿Cómo lo lograremos? ¿Qué tenemos?

Programaremos el cliente y servidor de una API.

Para esto deberemos **completar y corregir los siguientes componentes**:



DCCitas

Clase que se encarga de:

- **Mantener los estados de cada usuario**
- **Completar las solicitudes**

DCCitas

🅰️ path: str
🅰️ datos: dict
🅰️ tokens: dict

❗ cargar_datos() -> None
❗ guardar_datos() -> None

✅ registrar_usuario(nombre, password) -> None
✅ eliminar_usuario(nombre) -> None
✅ iniciar_sesion(nombre, password) -> token: str
✅ validar_token(nombre, token) -> None
✅ registrar_coordenadas(nombre, coordenadas) -> None
✅ encontrar_citas(nombre, cantidad) -> List[(nombre: str, distancia: float)]

🅰️ Atributo
✅ Método
❗ Con errores

API

Recibe las **solicitudes (*requests*)** del cliente y las procesa llamando los **métodos de DCCitas**.

Una vez procesada la solicitud, envía una **respuesta (*response*)** al cliente.

Algunas solicitudes necesitarán **autentificarse mediante un *token*** para validar al usuario y permitir que se realicen ciertas solicitudes.

Documentación API

POST `/registrar`

Request

Body

- `nombre: str`
- `password: str`

Response

201: Usuario registrado correctamente

```
{
  "msg": "Usuario registrado exitosamente"
}
```

Documentación API

DELETE `/eliminar/{nombre}`

Request

Parámetros

- `nombre: str`

Response

`204:` Usuario eliminado correctamente

```
{
  "msg": "Usuario eliminado exitosamente"
}
```

Documentación API

POST `/iniciar_sesion`

Request

Body

- `nombre: str`
- `password: str`

Response

`200`: Sesión iniciada exitosamente

```
{
  "msg": "Sesión iniciada exitosamente",
  "token": "5iY5JYmn7ALU"  # token asociado
}
```

`401`: Contraseña inválida

```
{
  "error": "Contraseña inválida"
}
```

Documentación API

PATCH `/registrar_coordenadas`

Request

Headers

- `Authorization: str`
  Token del usuario

Body

- `nombre: str`
- `coordenadas: list[floats]`

Response

200: Usuario registrado correctamente

```
{
  "msg": "Coordenadas registradas exitosamente"
}
```

401: Token inválido

```
{
  "error": "Token inválido"
}
```

Documentación API

GET `/encontrar_citas`

Request

Parámetros

- `n: int`
  Optativo, si no se entrega utiliza 1.

Headers

- `Authorization: str`
  Token del usuario

Body

- `nombre: str`

Response

`200:` Candidatos/as encontrados

```
{
  "msg": "Candidatos/as encontrados",
  "candidatos": [["Nombre", distancia], ...]
}
```

`401:` Token inválido

```
{
  "error": "Token inválido"
}
```

Parte 1: Cargar y guardar datos usuarios

DCCitas

`servidor/dccitas.py`

Los métodos de **`cargar_datos()`** y **`guardar_datos()`** no logran cargar y guardar la información de los usuarios correctamente.

Revisa cada método y corrígelo para que funciones como es esperado.

¡A programar! 💘

🅰️ `path: str`
🅰️ `datos: dict`
🅰️ `tokens: dict`

❗ `cargar_datos()`
❗ `guardar_datos()`

✅ `registrar_usuario()`
✅ `eliminar_usuario()`
✅ `iniciar_sesion()`
✅ `validar_token()`
✅ `registrar_coordenadas()`
✅ `encontrar_citas()`

Parte 2: Completar API

API

`servidor/main.py`

Los distintos ***endpoints* de la API se encuentran incompletos**. Revisa cada uno de ellos para que maneje correctamente cada solicitud.

Nota: En las diapos anteriores puedes revisar la documentación de la API.

¡A programar! 💘

POST	`/registrar`
DELETE	`/eliminar/{nombre}`
POST	`/iniciar_sesion`
PATCH	`/registrar_coordenadas`
GET	`/encontrar_citas`

Parte 3: Completar Cliente

Cliente

`cliente/main.py`

El cliente ya presenta los distintos métodos que **llaman a la API** de DCCitas, pero estos **están fallando**. Revisa cada uno y corrígelo.

🅰️ `base: str`
🅰️ `nombre: str`
🅰️ `token: str`

❗ `registrar_usuario(password: str) -> bool`
❗ `eliminar_usuario() -> bool`
❗ `iniciar_sesion(password: str) -> bool`
✅ `calcular_coordenadas() -> tuple`
❗ `registrar_coordenadas() -> bool`
❗ `encontrar_cita() -> bool`

¡A programar! 💘

Desafíos

Para el **cliente**:

- Actualmente las coordenadas de cada usuario se calculan al azar, pero podrían obtenerse a partir de algún quiz o información personal.

Para el **servidor**:

- Los *endpoints* están un poco desordenados: ¿Qué otras estructuras se podrían utilizar? ¿Se podrían unificar *endpoints*?
- Las contraseñas de los usuarios se guardar como texto plano: ¿Está bien eso? ¿Cómo lo hacen otras programas?

Programación Avanzada
IIC2233 2025-2

Cristian Ruz - Pablo Araneda - Francisca Ibarra - Tamara Vidal - Daniela Concha