TP2 > INTEGRATION

Flask + GWT + Net Core

Arquitecturas web

Objetivos

Integrar las 3 plataformas de chat de modo tal que:

- **Usuarios de diferentes plataformas puedan enviarse mensajes.**
- **Puedan armarse grupos entre usuarios de diferentes plataformas.**

Requerimiento no funcional:

- Priorizar que sea ‘*sencillo*’ para una 4ta Aplicación INTEGRARSE al sistema.

Esquema general

Client

GWT Server

Integration Server

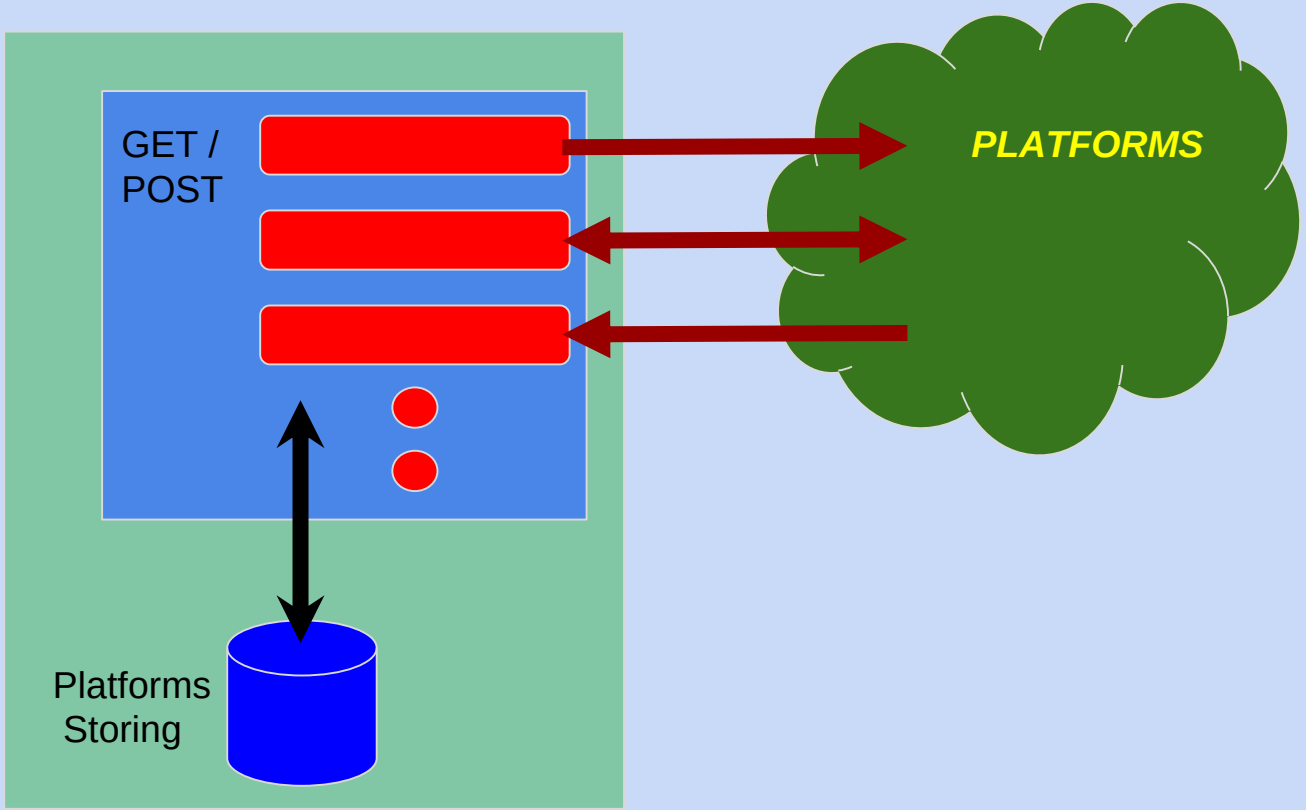
.Net Server

Client

Client

Flask Server

Integrador: Arquitectura



Integrador: Platform Storing

```
[
    {
        "id": 1,
        "name": "GWT_CHAT",
        "endpoint": "http://localhost:8080/api/",
        "supportAudio": "True",
        "supportImage": "True",
        "token": "1111"
    },
      ...
]
```

Integrador: ENDPOINTS

```
('/platforms', methods=['GET'])
('/user', methods=['POST'])
('/room', methods=['POST'])
('/message', methods=['POST'])

('/users', methods=['GET'])
('/ping', methods=['GET', 'POST'])
```

GET /platforms

Retorna

```
[
  {
    name :: String,
    supportAudio :: Boolean,
    supportImage :: Boolean,
    users :: [
        {
            id :: Integer,
            name :: String,
            platform :: String
        }
    ]
  },
        . . .
]
```

Ejemplo

```
[
  {
    'name': BuasApp,
    'supportAudio': 'True',
    'supportImage': 'True',
    'users': [
        {
            'id': 1,
            'name': 'Marcelo',
            'platform': 'BuasApp'
        }
    ]
  },
        . . .
]
```

POST /user

Requiere

```
{
    id :: Integer,
    name :: String,
    token :: String
 }
```

Retorna

```
{
    status: Int,
    message: String
}
```

Ejemplo

```
{
    'id' : 4,
    'name' : 'Juan',
    'token' : '4444'
 }
```

Se forwardea a las demás apps:

```
{
    'id' : 4,
    'name' : 'Juan',
    ‘platform’: 'BuasApp’'
 }
```

POST /room

Requiere

```
{
    id :: Integer,
    name :: String,
    token :: String,
    type :: Enum('private', 'public'),
    users :: [
                {
                    platformName:: [
                                userId
            ]
        }
    ]
}
```

Ejemplo

```
{
    'id' : 4,
    'name' : 'Los mundialistas',
    'token' : '4444'
    'type': public,
    'users': [
        {
                    "GWT_CHAT": [1,2,3]
        },
        {
                    "net core": [3,5,9]
        },
        {
                    "buatsaapp": [6,8]
        }
    ]
}
```

POST /message

Requiere

```
{
    roomOriginalPlatform :: String,
    roomId :: Integer,
    senderId :: Integer,
    text :: String,
    token :: String
}
```

Ejemplo

```
{
    'roomOriginalPlatform' : ‘BuasApp’,
    'roomId' : 4,
    'senderId': 3,
    'text': 'Este es un nuevo mensaje.',
    'token' : '4444',
}
```

Donde...

roomOriginalPlatform: *Nombre de la plataforma donde se creó el room.*
roomId: *Identificador del room al cual enviar el mensaje.*
senderId: *Identificador del usuario que generó el mensaje.*
text: *Mensaje a enviar.*
token: *Identificador de la app.*

Aclaración sobre el "roomId":

En las implementaciones de GWT y .Net el room id se corresponde a un room de web-socket, mientras que en la app del grupo de Flask el roomId se corresponde a una id de conversación la cual se muestra en cada room de cada usuario en esa conversación. Ampliaremos más adelante sobre esta diferencia de implementación y sus consecuencias derivadas.

Para poder integrarse toda app debe implementar:

```
@mod_api.route('/api/users', methods=['GET'])

@mod_api.route('/api/user', methods=['POST'])

@mod_api.route('/api/room', methods=['POST'])

@mod_api.route('/api/message', methods=['POST'])
```

Discusiones de integración:

La principal diferencia consistió en un modelo de simple forwardeo de mensajes vs un modelo centralizado que contenga una base de datos para los usuarios y los mensajes de todas las aplicaciones.

Finalmente prevaleció el modelo de forwardeo descentralizado (dos contra uno ;))

Discusiones de integración:

Otra discusión consistió en si debiéramos permitir o no que los usuarios pudieran loguearse en las otras aplicaciones.

Prevaleció que los usuarios solo se pueden loguear en sus aplicaciones de origen.

Discusiones de integración:

Sobre el protocolo:

Al momento de mandar un nuevo mensaje discutimos sobre qué parámetros era necesario enviar en el json: (solo el id del room vs agregar también el id de los usuarios involucrados).

Esto se debió a diferencias internas de implementacion sobre los web sockets: (próxima diapo)

Discusiones de integración:

Los grupos de .Net y GWT implementaron un room por conversación, mientras que en el grupo de BuasApp implementamos un room por usuario. Esto significa que al momento de importar un mensaje externo a todos nuestros usuarios de BuasApp necesitamos buscar en la base cuales son todos esos usuarios para saber a cuales rooms debemos mandar el mensaje. Esto impacta en nuestra performance.

Discusiones de integración:

Si el json del mensaje nuevo especificará los usuarios involucrados y no solamente el id del room la performance en la aplicación en Flask mejoraría pero probablemente impactaría de forma contraria en las otras apps.

Implementación en BuasApp:

Nuevo módulo integrador:

Controller: contiene los endpoints que son alcanzados desde la api integradora.

Importer: Se encarga de guardar en la base todo y de forwardear los mensajes al internal messenger.

Implementación en BuasApp:

Nuevo módulo integrador:

Exporter: es el encargado de enviar a la api integradora los eventos internos que necesitamos exportar.

Internal messenger: Envía a nuestros clientes (por web sockets) los mensajes que llegan desde la api integradora.

Implementación en BuasApp:

Cambios en el modelo de datos:

- **Se agregó una nueva tabla de plataformas.**
- **Se agregaron dos columnas tanto en la tabla de conversaciones como en la de usuarios:**

    1. **External_id (string)**
    2. **Platform_id (int - foreign key a la tabla de plataformas)**

Implementación en GWT:

Nuevos módulos:

- **IntegrationServlet: Se encarga de recibir la información de la App Integradora.**

Implementación en GWT:

Nuevos módulos:

- **IntegrationClient: Es utilizada para enviar a la App Integradora la información local ya traducida al formato adecuado.**

Implementación en GWT:

Nuevos módulos:

- **IntegrationService: Es un conjunto de funcionalidades centradas en:**
  - **Actualizar la base de datos ante el ingreso de información de parte de la App Integradora.**
  - **Dar formato a la información local que, posteriormente, recibirá la App Integradora.**

Implementación en GWT:

Cambios en el modelo de datos:

- **Se cambió el identificador de la tabla conversaciones de un entero a un string con la finalidad de poder construir los identificadores de los rooms externos como la concatenación del nombre de la plataforma, un símbolo el cual no puede aparecer como parte de un id en nuestra base y el id original del room externo. Realizamos el mismo proceso con los usuarios.**

Para la próxima:

Crear un proceso batch que importe a nuestra app todos los usuarios externos llamando a la api integradora cada X cantidad de tiempo. Posiblemente una vez por día.

Guardar los mensajes que se pierden cuando alguna app está caída y forwardearlos cuando se reconecta.