

Índice de contenido

Documentación de la API Rest.....	2
Referencias.....	2
Nodos raíz.....	2
Escritura en nodos.....	2
Autenticando.....	2
Parametros.....	2
Respuesta.....	2
Desautenticación.....	3
Parametros.....	3
Respuesta.....	3
Conversaciones.....	3
Usuarios.....	3
Perfil.....	4
Avatar.....	5
Checkin.....	6
Mensajes.....	7
Notificaciones.....	9
Lectura.....	9
Parámetros.....	9
Respuesta.....	9
type=message.....	10
type=ack.....	10
Escritura.....	11
Parámetros.....	11
Respuesta.....	11
Registro.....	11
Parámetros.....	11
Respuesta.....	12
Manejo de errores.....	12
Códigos de estado de HTTP.....	12
Respuestas de error.....	13
ChangeLog.....	13
División de tareas y seguimiento.....	24
1er Milestone: Funcionalidades v0.1.....	24
Planificación de tareas para 1er CheckPoint.....	24
Separación de tareas.....	25
Documentación de código.....	25

Informe 1^{er} CheckPoint

Documentación de la API Rest

Referencias

Nodos raíz

- /auth: Autenticación
- /conversation: Conversaciones
- /user: Usuarios
- /notification: Notificaciones
- /singup: Registro

Escritura en nodos

La escritura a los nodos mediante POST se podrá hacer usando tanto application/x-www-form-urlencoded como multipart/form-data. Se recomienda solo usar multipart/form-data cuando se envíe data binaria como por ejemplo imágenes. Preferir la otra disposición de contenido.

Autenticando

```
POST /auth HTTP/1.1
```

```
Host: <HOST>
```

```
user=username&pass=password
```

Parametros

- user : Nombre de usuario
- pass : Contraseña

Respuesta

```
HTTP/1.1 201 CREATED
```

```
{
```

```
  "access_token": [string] token de acceso a usar en la api
```

```
}
```

Desautenticación

Parametros

```
DELETE /auth/{access_token} HTTP/1.1  
Host: <HOST>
```

Respuesta

```
HTTP/1.1 200 OK  
{  
  "success": true  
}
```

Conversaciones

Nodo relacionado a las conversaciones (grupales). Desde este nodo y sus subnodos se realizaran todas las acciones relacionadas con las conversaciones, ya sea, crearlas, agregar/quitar usuarios, obtener mensajes, información, etc.

El nodo `/conversation/{conversation-id}` representa una conversación.

Las conversaciones de grupo no son algo que esta puesto en el enunciado. Si hay tiempo habría que definirlas e implementarlas.

Usuarios

Desde este nodo y sus subnodos se realizaran todas las acciones relacionadas con los usuarios, ya sea, pedido de estado, envío de mensajes.

- `/user/{user-name}`: Representa a un usuario.
- `/user/me`: Representa al usuario conectado, es un alias, a `/user/{user-name}` utilizando el `{user-name}` del usuario actual.

Nota: Todas las llamadas requieren de su `access_token` correspondiente!.

Existen los siguientes subnodos:

- /user/{user-name}/profile: [Perfil](#)
- /user/{user-name}/avatar: [Avatar](#)
- /user/{user-name}/checkin: [Checkin](#)
- /user/{user-name}/messages: [Mensajes](#)

Perfil

/user/{user-name}/profile

Lectura

GET /user/{user-name}/profile HTTP/1.1

Host: {HOST}

Respuesta

HTTP/1.1 200 OK

```
{
  "username": [string] Username
  "nickname": [string] Nombre
  "online": [bool] Conectado / desconectado
  "last_activity": [epoch] fecha de última interacción con el servidor
  "status": {
    "time": [epoch] fecha de actualización
    "text": [string] Texto del estado
  }
  "checkin": {
    "time": [epoch] fecha de actualización
    "latitude": [number] Latitud
    "longitude": [number] Longitud
    "name": [string] Nombre del lugar
  }
}
```

Escritura

Este nodo solo puede ser accedido en modo escritura por el usuario al que pertenezca el nodo.

POST /user/{user-name}/profile HTTP/1.1

Host: {HOST}

nickname=apodo&status=mi+estado

Parámetros

- nickname: [string] opcional Apodo a usar
- online: [bool] opcional true para conectado, false para desconectado
- status: [string] opcional Estado del usuario

Respuesta

HTTP/1.1 201 CREATED

```
{  
  "success": true  
}
```

Avatar

/user/{user-name}/avatar

Lectura

GET /user/{user-name}/avatar HTTP/1.1

Host: {HOST}

Retorno

HTTP/1.1 200 OK

{IMAGEN}

Escritura

Este nodo solo puede ser accedido en modo escritura por el usuario al que pertenezca el nodo. La imagen se deberá mandar en POST usando multipart/form-

data.

POST /user/{user-name}/avatar HTTP/1.1
Host: {HOST}

Parámetros

- avatar: Imágen a usar de avatar

Respuesta

```
HTTP/1.1 201 CREATED
{
  "success": true
}
```

Checkin

/user/{user-name}/checkin

Lectura

GET /user/{user-name}/checkin HTTP/1.1
Host: {HOST}

Respuesta

```
HTTP/1.1 200 OK
{
  "time": [epoch] fecha de actualización
  "latitude": [number] Latitud
  "longitude": [number] Longitud
  "name": [string] Nombre del lugar
}
```

Escritura

Este nodo solo puede ser accedido en modo escritura por el usuario al que pertenezca el nodo.

POST /user/{user-name}/checkin HTTP/1.1

Host: {HOST}

latitude=22.5&longitude=44.1&name=Casa+de+Juan

Parámetros

- latitude: [number] latitud
- longitude: [number] longitud
- name: [string] opcional Nombre del lugar

Respuesta

HTTP/1.1 201 CREATED

```
{  
  "success": true  
}
```

Mensajes

/user/{user-name}/messages

Lectura

GET /user/{user-name}/messages HTTP/1.1

Host: {HOST}

Parámetros

- limit: [number] opcional Cantidad máxima de mensajes a recibir
(Defecto: 20)
- last_id: [string] opcional Traer mensajes anteriores al identificador especificado

Respuesta

HTTP/1.1 200 OK

```
{
```

```
"next": [string] Link a siguientes mensajes,  
"messages": [  
  {  
    "id": [string] Identificador del mensaje  
    "from": [string] Username de quien mandó el mensaje  
    "time": [epoch] Fecha del mensaje  
    "arrived": [epoch] Hora de cuando le llego al destino (o 0 si no llego)  
    "read": [epoch] Hora de cuando fue leído (o 0 si no lo fue)  
    "message": [string] Mensaje  
    "file": [string] Url a un archivo relacionado con el mensaje  
    "file_type": [string] MIME type del archivo  
  }  
  ...  
]  
}
```

El orden de la lista de mensajes es en time decreciente. El primero de la lista es un mensaje cuyo time es menor o igual al segundo y así sucesivamente. Es decir, los mensajes van del más nuevo al más viejo.

Escritura

Este nodo se usará para enviar mensajes al usuario.

POST /user/{user-name}/messages HTTP/1.1

Host: {HOST}

message=este+es+un+lindo+mensaje&random_id=456789411254785

Cuando se envíen archivos binarios, se deberá mandar
en POST usando multipart/form-data

Parámetros

- message: [string] Mensaje a enviar
- random_id: [string] Identificador aleatorio usado para prevenir el reenvío de

mensajes.

- file: [binary] opcional Archivo a enviar
- file_type: [string] opcional MIME type del archivo enviado

Respuesta

HTTP/1.1 201 CREATED

```
{
  "id": [string] Identificador del mensaje
  "time": [epoch] Fecha del mensaje
}
```

Notificaciones

/notification

Lectura

GET /notification HTTP/1.1

Host: {HOST}

Parámetros

Sin parámetros

Respuesta

HTTP/1.1 200 OK

```
{
  "notifications": [
    {
      "id": [string] id de la notificación
      "type": [string] Tipo de notificación
      "time": [epoch] tiempo de la notificación
      "data": {
        ...
      }
    }
  ]
}
```

```
]
}
```

El orden de la lista de mensajes es en time creciente. El primero de la lista es una notificación cuyo time es mayor o igual al segundo y así sucesivamente. Es decir, las notificaciones van de la más vieja a la más nueva.

El type representa el tipo de notificación, el data representa la información.

Los types posibles son:

- message: Para mensajes
- ack: Para arribo o lectura de mensajes
- TODO

type=message

```
{
  "id": [int] id de la notificación
  "type" : "message"
  "time": [epoch] tiempo de la notificación
  "data": {
    "id": [string] Identificador del mensaje
    "from": [string] Username de quien mandó el mensaje
    "time": [epoch] Fecha del mensaje
    "arrived": [epoch] Hora de cuando le llego al destino (o 0 si no llego)
    "read": [epoch] Hora de cuando fue leído (o 0 si no lo fue)
    "message": [string] Mensaje
    "file": [string] Url a un archivo relacionado con el mensaje
    "file_type": [string] MIME type del archivo
  }
}
```

El contenido de data es el mismo que el de un [Mensaje](#).

type=ack

```
{
  "id": [int] id de la notificación
  "type" : "ack"
  "time": [epoch] tiempo de la notificación
  "data": {
```

```
"id": [string] Identificador del mensaje
"from": [string] Username de quien mandó el mensaje
"time": [epoch] Fecha del mensaje
"arrived": [epoch] Hora de cuando le llegó al destino (o 0 si no llegó)
"read": [epoch] Hora de cuando fue leído (o 0 si no lo fue)
}
}
```

Escritura

Este nodo se usará para marcar que las notificaciones llegaron.

```
DELETE /notification?id=123456789 HTTP/1.1
Host: {HOST}
```

Parámetros

- `id`: [string] Especifica hasta que notificación se recibió (es incluyente).

Respuesta

```
HTTP/1.1 200 OK
{
  "success": true
}
```

Registro

Nodo relacionado con el registro de un usuario al sistema.

```
POST /signup HTTP/1.1
Host: <HOST>

user=username&pass=password
```

Parámetros

- `user` : Nombre de usuario
- `pass` : Contraseña

El campo `user` debe ser una cadena de caracteres de mayor largo que 0. Los caracteres validos matchean la siguiente expresión regular: `[0-`

9a-z,._-]* además, son case insensitive.

El campo pass debe ser una cadena de caracteres de mayor largo que 6.

Respuesta

```
HTTP/1.1 201 CREATED
```

```
{  
  "success": true  
}
```

Manejo de errores

Los pedidos hechos a la API pueden resultar en diferentes respuestas de error. Aquí se describe la lista de valores de error a esperar.

Códigos de estado de HTTP

Las respuestas a la API devolverán los siguientes códigos de estado para informar errores:

- 400: Cuando se hace un pedido a la API y los parámetros especificados no cumplen las precondiciones pedidas.
- 401: Cuando se intenta acceder a contenido que requiere autenticación sin suministrar un 'access_token' o con uno inválido.
- 403: Cuando se intenta acceder a contenido que el usuario del 'access_token' suministrado no puede ver.
- 404: Cuando se intenta acceder a contenido que no existe.
- 405: Cuando se hace un pedido a un endpoint con un método de HTTP que no corresponde. Por ej: le pedí a un endpoint por GET y este no lo implementa.
- 500: Error interno del servidor, cualquier tipo de problema que es ajeno a los datos del cliente y es causa de problemas internos del servidor.

Respuestas de error

```
{
  "error": {
    "message": [string] Mensaje describiendo el error para el desarrollador,
    "code": [number] Código de error,
    "error_user_msg": [string] Mensaje de error para el usuario
  }
}
```

ChangeLog

	DISEÑO DE API REST
	Mensajer0
15/03/2015	Nicolás
	<p>Links:</p> <ul style="list-style-type: none">•Introduccion rest•Json api <p>Nota respecto al login: Como REST es stateless en todas las consultas que requieran una autenticación se tiene que enviar la información necesaria para la misma (en un principio seria user + pass). El enunciado habla de generar una sesión de usuario, técnicamente no seria una sesión si no generar un "access token"</p> <p>Leer:</p> <ul style="list-style-type: none">•Rest api login pattern Stackoverflow•Do session really violate RESTfullness ? Stackoverflow•How do i let users log into my RESTfull api
20/03/2015	Nicolás
	<p>Este archivo tiene la info acerca de como va a ser el protocolo. Cosas a corregir:</p> <ul style="list-style-type: none">•Se debe hacer una ruta para login, va a ser donde se envíe user y pass para generar el access_token que se usaran en todas las consultas•Como el servidor solo va a proveer el servicio de whatsapp, no hace falta que utilicemos la subrutawhatsapp para todas los pedidos•El pedido de la información de usuario NUNCA DEBERIA DEVOLVER LA PASSWORD•Las estructuras Json no deberian devolver el codigo de error ya que este ya se encuentra en el request de

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	<p>http.</p> <ul style="list-style-type: none"> •Falta definir básicamente todas las rutas con sus respectivos parámetros <p>Nota: usar los siguientes codes de http</p> <ul style="list-style-type: none"> •200: Correcto •401: Falta de login (no se mando el access_token) •403: Estas conectado, pero quieres acceder a algo qe no podes ver (hay algún caso?) •404: Pusiste cualquier url <p>Ejemplo Login POST /login Parámetros:</p> <ul style="list-style-type: none"> •user: usuario •pass: contrase~na <p>Rtas Correcta code: 200 { "access_token": "asdasdasd" }</p> <p>Incorrecta code: 401 { "message": "invalid credentials" }</p>
07/04/2015	Nicolas
	<p>Se diseño el primer boceto de la api REST https://github.com/NickCis/7552-taller-prog-2-2015-C1/wiki/Referencia</p>

	Estructura básica del servidor
	Mensajer0
15/03/2015	Nicolás
	<p>Ver si nos permiten usar las siguientes librerías:</p> <ul style="list-style-type: none"> •libuv •http-parser <p>Resolver como el servidor maneja connecciones:</p> <ul style="list-style-type: none"> •Usando libuv con i/o asincronico •Lanzando un thread por cada conecion •Usando un thread pool con una cantidad finita de threads <p>Pensar esquema para desarrollar de manera modular todas las url del esquema REST</p>
19/03/2015	Nicolás
	Nos piden usar mongoose

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

08/04/2015	Nicolás
	Se sube primer prototipo de clases para los nodos
11/04/2015	Nicolás
	Se hacen prototipos para la estructura básica del servidor. Abstracción de librería mongoose y clases bases para nodos
	Se mueven archivos para que las carpetas queden mas coherentes. Se agrega archivos vim a gitignore
	Se modifican paths de archivos
	Se agrega WAMethodNode un nodo que provee metodos para el manejo de los methods de HTTP
	Se hace prototipo para sacar parámetro de usuario de la url y guardarlo en la coneccion
	Se agregan métodos para obtener parámetros get/post

	Decidir framework para tests
	Mensajer0
15/03/2015	Nicolás
	Cual se les ocurre?
26/03/2015	Martin
	Propongo CUTE.
28/03/2015	Nicolás
	Vamos a usar este: https://github.com/cpptest/cpptest
	Voy a usar <code>catch</code> por que solo hay que incluir un <code>.h</code>
	Se sube mock de como realizar tests. Se agrega al travis ejecutarlos

	Definir notificaciones pull
--	-----------------------------

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	--

	Mensajer0
07/03/2015	Nicolás
	Faltaría definir como la API para que se pulen las notificaciones Nicolás
17/04/2015	Las notificaciones pull quedaron definidas en la wiki

	Definir interfaz usuario
	Mensajer0
07/03/2015	Rodrigo
	Definir el look and feel del cliente Estructura basica
25/04/2015	Rodrigo
	Se termina #2 estructura básica del cliente parte de #11 interfaz de usuario. ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (próximos commits) tenemos ip configurable, se cierra #37 También agregamos pantalla de registraron, #38 terminado cerrado #39 cuadros de dialogo informativos
	parte de #11 interfaz de usuario. closes #11 ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (próximos commits). closes #31 tenemos ip configurable, se cierra #37. closes #37 También agregamos pantalla de registracion, #38 terminado . closes #38 cerrado #39 cuadros de dialogo informativos. closes #39

	Desarrollar endpoint SignUp
	Mensajer0
11/04/2015	Nicolas
15/04/2015	Nicolas
	Se corrigen parámetros de /signup endpoint

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	Se sube Endpoint /auth, ya se manejan los access_tokens. Se arregla problema de Status codes para /auth y /signup
--	---

	Desarrollar endpoint /user/{username}/messages
	Mensajer0
11/04/2015	Nicolás
21/04/2015	Nicolás
	Auth: se hacen cambios, métodos estáticos con mayúscula. #15 Mensajes: se agrega clase de db y endpoint.
	Ya se pueden recibir mensajes. No se controla que receptor exista, no se despachan notificaciones :)
	Los mensajes incluyen mas campos
22/04/2015	Nicolás
	Se remueve campo to en la clase Message
	Message: Se usa el retorno del Pack

	Definir logging
	Mensajer0
11/04/2015	Nicolás
	Definir una herramienta para loggear, la sintaxis podría ser parecida a la que usa qDebug() (se puede usar de base esta: .h y .cpp Recordar que el enunciado pide estos niveles de log: •Error

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	<ul style="list-style-type: none"> •Warn •Info •Debug <p>Estaría bueno que cuando arranca el servidor loggee cuando fue compilado, la versión, versión de mongoose, y hash de revisión del git.</p>
26/04/2015	Martín
	Propongo log4cpp para el server y log4J para el cliente.
27/04/2015	Nicolás
	se implementa clase para logear

	Incorporar RocksDB a MgServer
	Mensajer0
11/04/2015	Nicolás
	Recordar que RocksDb es threadsafe. Ver como se va a guardar la información en rocksdb (recordar que es una db que lo único que permite es guardar valores key-value)
13/04/2015	Nicolás
	Incorporar RocksDB a MgServer y que sea accesible via MgConnection to Incorporar RocksDB a MgServer
	Se agrega rocksdb a CMake
	Se agregan clases para manejar la db y crear usuarios
15/04/2015	Nicolás
	Se corrigen parámetros de /signup endpoint

	Servidor tira Segmentation Fault al salir
	Mensajer0
15/04/2015	Nicolás
	Fijarse por que el servidor esta tirando un segmentation fault al salir.

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	Crear una estructura de datos para soportar almacenamiento parcial
	Mensajer0
17/04/2015	Matias
	Crear una estructura de datos interna para almacenar en el dispositivo conversaciones activas y mensajes de las mismas hasta cierto punto para un rápido acceso a ellas. El resto del historial será cargado del servidor a pedido.
	Creada una estructura interna basada en Android SQLITE para almacenamiento parcial de conversaciones activas. Issue #22. Falta acoplarla a las activities

	MgConnection::getVarStr agrega fruta a los valores
	Mensajer0
21/04/2015	Nicolás
	Esta devolviendo un string con largo 64 Se arregla problema con MgConnection::getVarStr

	Implementar notificaciones
	Mensajer0
22/04/2015	Nicolás
	Cuando llega un mensaje debería generar la notificación. Además, se debería implementar el endpoint de notificaciones
	Notificaciones: Se agrega clase de db que implementa notificaciones. Compila.

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	Hay que usar el comparator de mensajes para tener un orden similar. Para borrar se puede seekear obtener las ids y dsp borrar en batch
26/04/2015	Nicolás
	Arreglos en db manager para seleccionar el comparator de una manera mas comoda. #26 Notificaciones usan un comparator especifico
	Se agrega endpoint notification, mensajes deberían reportar las notificaciones. Falta marcar notificaciones como leídas
	DbManager setea la db y column family a Notification
	RootNode incluye NotificationNode
	Andan notificaciones a mensajes. Falta marcar notificaciones como leídas
	Se pueden borrar notificaciones

	CheckIn usuario
	Mensajer0
22/04/2015	Rodrigo
	Del enunciado : Checkin Desde el cliente el usuario podrá confirmar su presencia (checkin) en un lugar. Para obtener la posición del cliente se deberá poder utilizar la ubicación del dispositivo [2]. Este checking debe ser enviado al servidor para que surta efecto.
25/04/2015	Rodrigo
	se termina #2 estructura basica del cliente parte de #11 interfaz de usuario. ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (próximos commits) tenemos ip configurable, se cierra #37 Tambien agregamos pantalla de registracion, #38 terminado cerrado #39 cuadros de dialogo informativos parte de #11 interfaz de usuario. closes #11 ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (proximos commits). closes #31 tenemos ip configurable, se cierra #37. closes #37 Tambien agregamos pantalla de registracion, #38 terminado . closes #38 cerrado #39 cuadros de dialogo informativos. closes #39
	parte de #11 interfaz de usuario. closes #11 ahora tenemos #31 checkin de usuario, del lado de la vista,

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	falta conectarse con servidor (próximos commits). closes #31 tenemos ip configurable, se cierra #37. closes #37 Tambien agregamos pantalla de registraron, #38 terminado . closes #38 cerrado #39 cuadros de dialogo informativos. closes #39

	Mensajes: Anda mal ordenado y filtrado
	Mensajer0
23/04/2015	Nicolas
	Basicamente esta andando mal el endpoint de mensajes,, Se desarrollo un Comparator para ordenar la column family de los mensajes de una manera que sea fácil obtener los últimos mensajes y los de una misma conversación. Aparentemente esta andando bien lo de obtener los de una misma conversación, pero no en el orden, parece que hay un problema con el id de los mensajaes
	Mensajes: se sube el comparator y un script para generar un caso de prueba.
	Se usa de id el tiempo en usecs en big endian, eso arregla el orden de las entradas en la db
	Se arregla problema con id mal escrita en el JSON
24/04/2015	Nicolás
	Mensajes: iterador ignora el mensaje cabecera que guarda el id del ultimo mensaje
	Mensajes: iterador ignora el mensaje cabecera que guarda el id del ultimo mensaje Mensajes devuelve los últimos
	Se agrega limit a nodo mensajes
	Mensajes ya se pueden recibir correctamente, devuelve de manera ordenada, acepta parámetros limit y last_id

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	--

	Ip configurable
	Mensajer0
25/04/2015	Rodrigo
	El cliente puede elegir ip del servidor a donde hacer pedidos
	se termina #2 estructura basica del cliente parte de #11 interfaz de usuario. ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (proximos commits) tenemos ip configurable, se cierra #37 Tambien agregamos pantalla de registracion, #38 terminado cerrado #39 cuadros de dialogo informativos
	parte de #11 interfaz de usuario. closes #11 ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (proximos commits). closes #31 tenemos ip configurable, se cierra #37. closes #37 Tambien agregamos pantalla de registracion, #38 terminado . closes #38 cerrado #39 cuadros de dialogo informativos. closes #39

	Pantalla de registro
	Mensajer0
25/04/2015	Rodrigo
	Se distinguen pantallas de registro y login
	se termina #2 estructura basica del cliente parte de #11 interfaz de usuario. ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (proximos commits) tenemos ip configurable, se cierra #37 Tambien agregamos pantalla de registracion, #38 terminado cerrado #39 cuadros de dialogo informativos
	parte de #11 interfaz de usuario. closes #11 ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (proximos commits). closes #31 tenemos ip configurable, se cierra #37. closes #37 Tambien agregamos pantalla de registracion, #38 terminado . closes #38 cerrado #39 cuadros de dialogo informativos. closes #39

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	Diálogos informativos
	Mensajer0
25/04/2015	Rodrigo
	Informar a usuario de las cosas q estan pasando en segundo plano
	se termina #2 estructura basica del cliente parte de #11 interfaz de usuario. ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (próximos commits) tenemos ip configurable, se cierra #37 También agregamos pantalla de registracion, #38 terminado cerrado #39 cuadros de dialogo informativos
	parte de #11 interfaz de usuario. closes #11 ahora tenemos #31 checkin de usuario, del lado de la vista, falta conectarse con servidor (próximos commits). closes #31 tenemos ip configurable, se cierra #37. closes #37 También agregamos pantalla de registracion, #38 terminado . closes #38 cerrado #39 cuadros de dialogo informativos. closes #39

	Notificaciones están desordenadas
	Mensajer0
26/04/2015	Nicolás
	Las notificaciones vienen desordenadas, chequear por que. Agregar validación de información a las notificaciones
26/04/2015	Nicolás
	DbComparator, castea chars a unsigned char para que la comparacion sea correcta

1º Cuatrimestre 2015 Trabajo práctico - Mensajero	Taller de Programación II (75.52) Facultad de Ingeniería Universidad de Buenos Aires
--	---

	Informe 1er CheckPoint
	Mensajer0
26/04/2015	Martín
	Realizar informe incluyendo. - División y planificación de tareas. - Documentación de código - Evidencia (tickets)

División de tareas y seguimiento

A continuación se listan la funcionalidades que se pusieron como objetivo para el primer checkpoint

1^{er} Milestone: Funcionalidades v0.1

- Crear usuarios
- Login de usuarios
- Intercambio de mensajes
- Notificaciones de mensajes nuevos

Planificación de tareas para 1er CheckPoint.

	Semana							
Actividades	1	2	3	4	5	6	7	8
Análisis funcional								
Diseño								
Codificación								
--Server								
--Client								
Documentación								
Test								

Separación de tareas

	Grupo de desarrollo			
Actividades	Nicolás	Rodrigo	Matias	Martín
Análisis funcional	x	x	x	x
Diseño	x	x	x	x
Codificación				
--Server	x			x
--Client		x	x	
Documentación				x
Test	x	x	x	x

Documentación de código.

La documentación de código se realizó usando la herramienta doxygen, con la que se generó un documento HTML. Esta está separada en dos partes, la parte del cliente y la del servidor.

La parte del cliente se puede encontrar en ../client/doc, y la parte del servidor en ../server/doc

Se dejó una screen de la pantalla principal.

MensajerO 0.1

Página principal

Páginas relacionadas




Clases

Archivos

Lista de archivos

Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

▶  db	
▶  mg	
▶  node	
▶  rest	
▶  util	
 main.cpp	
 wa_server.cpp	
 wa_server.h	