

Design Document for:

goChron

Written by

Nagapandu Potti
Aman Singh
Shantanu Kande
Akshitha Ainapuri
Rakesh Dammalapati
Jithendra Yella

Version # 1.00

September 18 2015

Table of Contents

GoChron

1.0 Introduction

1.1 Goals and objectives

1.2 Statement of scope

1.3 Software context

1.4 Major constraints

2.0 Data design

2.1 Internal software data structure

2.2 Global data structure

2.3 Temporary data structure

2.4 Database description

3.0 Architectural and component-level design

3.1 System Structure

3.1.1 Architecture diagram

3.2 User Class

3.2.1 Processing narrative (PSPEC)

3.2.2 Interface description.

3.2.3 Processing Detail

3.2.3.1 Class Hierarchy

3.2.3.2 Restrictions/limitations

3.2.3.4 Design constraints

3.2.3.5 Processing details

3.3 Event Class

3.3.1 Processing narrative (PSPEC)

3.3.2 Interface description.

3.3.3 Processing Detail

3.3.3.1 Class Hierarchy

3.3.3.2 Restrictions/limitations

3.3.3.3 Design constraints

3.3.3.4 Processing details

3.4 Group Class

3.4.1 Processing narrative (PSPEC)

3.4.2 Interface description.

3.4.3 Processing Detail

3.4.3.1 Class Hierarchy

3.4.3.2 Restrictions/limitations

3.4.3.3 Design constraints

3.4.3.4 Processing details

Software Design Document

3.5 Dynamic Behavior for Components

3.5.1 Interaction Diagrams

4.0 User interface design

4.1 Description of the user interface

4.1.1 Screen images

4.2 Interface design rules

4.3 Components available

5.0 Restrictions, limitations, and constraints

6.0 Testing Issues

Different types of testing will be performed to ensure smooth functionality of the product. Starting with unit test cases, each functionality will be tested as soon as the development of that particular module is finished. Integration testing would be performed after every module is integrated.

6.1 Classes of tests

6.2 Performance bounds

7.0 Appendices

7.1 Packaging and installation issues

7.2 Design metrics to be used

7.3 Supplementary information

Software Design Document

Revision History

Name	Date	Reason For Changes	Version
goChron SDD	09/18/15	NA	1.0

1.0 Introduction

The purpose of this software design document is to provide a low level description about goChron that includes information about the database design, structure of each component and the user interface going to be developed.

Topics covered include the following:

- Data Design
- Class hierarchies and interactions
- Design limitations and constraints
- User Interface Design
- Design metrics used

Briefly, this document is meant to facilitate the user with a solid understanding about the functionality of goChron.

1.1 Goals and objectives

The purpose of goChron is to simplify the way students keep track of events happening at UF by bringing onto a single platform with much enhanced features such as subscriptions, suggestions, alerts, notifications and currently trending events. It also provides an option for the user while creating an event to choose between public, private and closed.

Accordingly, the final product must be extremely easy to use and subscribe. It must offer useful features without overwhelming the user with options. The user interface must be intuitive and should have minimal learning curve. Beyond these general design principles, the application must also provide concrete functionalities like support for subscribing to multiple groups, automatic data synchronization and synchronizing with a third party calendar.

1.2 Statement of scope

goChron is composed of a client-side application that receives user input and also composed of a server-side application which updates and synchronizes the data regarding events across the systems.

The application has core features like creating events, subscribing to events, notifications, sending and receiving invitations which are essential for the functionality of the product and other additional features like synchronizing with other calendar, trending events which add extra functionality to the product. Once the basic functionality as described above is built within the given time frame, goChron might be enhanced to be a native mobile application.

The following list includes the list of core and additional features:

- CORE FEATURES:
 - User Registration and Login
 - Add/Reschedule an Event
 - Join/Drop an Event
 - Subscription
 - Alerts

- **ADDITIONAL FEATURES:**
 - Group Creation and Management
 - Synchronize with Other calendar

1.3 Software context

goChron will be available as a web application and it is free of charge. Costs for developing and maintaining goChron are virtually nonexistent, so funding is not a problem. However, if the situation changes and it requires funding in the future, it will be possible by incorporating on-screen advertisements of local firms in Gainesville.

All the software specifications and the UI design requirements regarding the basic functionality of goChron are specified in detail in the later sections of this software design document.

1.4 Major constraints

The greatest constraint of this goChron application is time. There is very less time allocated for development and testing this product. As most of the team members have little experience in developing a web application, major time will be allocated in understanding and learning the environment. Consequently, this may result in fewer features in the initial release, however the core functionality of goChron will be unaffected.

2.0 Data design

2.1 Internal software data structure

goChron's internal structure comprises of two parts: server-side and client-side. At the client side, the request from the web page is sent to the server for information which in fact communicates with the database. As the goChron application is based on user friendly GUI, in order to increase the performance or to be more responsive, the information that is needed more frequently will be stored using the n memory caching (redis) on RAM. Overall, the data is accessed by way of a Model-View Controller system.

The server will be implemented using Ruby on Rails. Permanent storage of user information will be accomplished using a NoSQL database such as Mongo db. The server and the client will exchange data using the JSON format. JSON is a lightweight object description language that is similar to XML.

2.2 Global data structure

The global data structure of this application is best characterized by the database. The database structure shows the data involved in the application. The client of goChron will never access this database directly; it will instead issue requests to the server.

2.3 Temporary data structure

Temporary data structures, in the perspective of goChron, refer to the data objects that are created such as JSON objects that are interchanged between the server and the client. The data objects created on the local device will only exist for the duration of time that the application is running, and will subsequently be destroyed.

The JSON objects will only exist for the duration of the transaction between the client and server. The server will destroy the objects after sending them, and the client will destroy the JSON objects once they have been parsed.

2.4 Database description

MongoDB is the back-end database for the application. This is a schema less (NoSQL) database, which means there are no fixed fields and clearly no field types for the collections (or tables).

Create Database:

```
> use gochron
```

Create Collections:

users collection :

```
> db.createCollection("users")
```

```
User {  
  field :name, type: String  
  field :email, type: DateTime  
  field :password_hash, type: String  
  field :password_salt, type: String  
  field : group_ids, type: Array  
}
```

events collection :

```
> db.createCollection("events")
```

```
Event {
```

```
field :title, type: String
field :datetime, type: DateTime
field :description, type: String
field :location, type: String
field :link, type: String
field :access, type: String
field :user_id, type: String /* creator of the event */
field :group_id, type: String
field :attendee_ids, type: Array /* Users attending this event */
}
```

groups collection :

```
> db.createCollection("groups")

Group {
field :name, type: String
field : title, type: String
field : description, type: String
field :user_id, type: String /* group admin */
field :member_ids, type: Array /* group members */
}
```


3.0 Architectural and component-level design

A description of the program architecture is presented.

3.1 System Structure

The goChron system comprises of two major components: a client-side application and a server-side ruby on rails application and Mongodb database.

The client-side application is split into two parts: the functional component (written in Javascript), and the graphical component (written in html, css, bootstrap). The functional component forms the core of goChron. It makes sure that the web page is dynamically updated based on the users activity. The graphical user interface presents a view in which it allows the user to access all of the features provided by the goChron application.

The server component of goChron comprises of Ruby on Rails Server, which manages incoming and outgoing messages, and a Mongodb database, which provides centralized storage for synchronized data. The server application receives data(JSON) from client nodes and sends it into Mongodb for useful information. This data is then stored in the database and subsequently synchronized with the UI at the client side.

3.1.1 Architecture Diagram

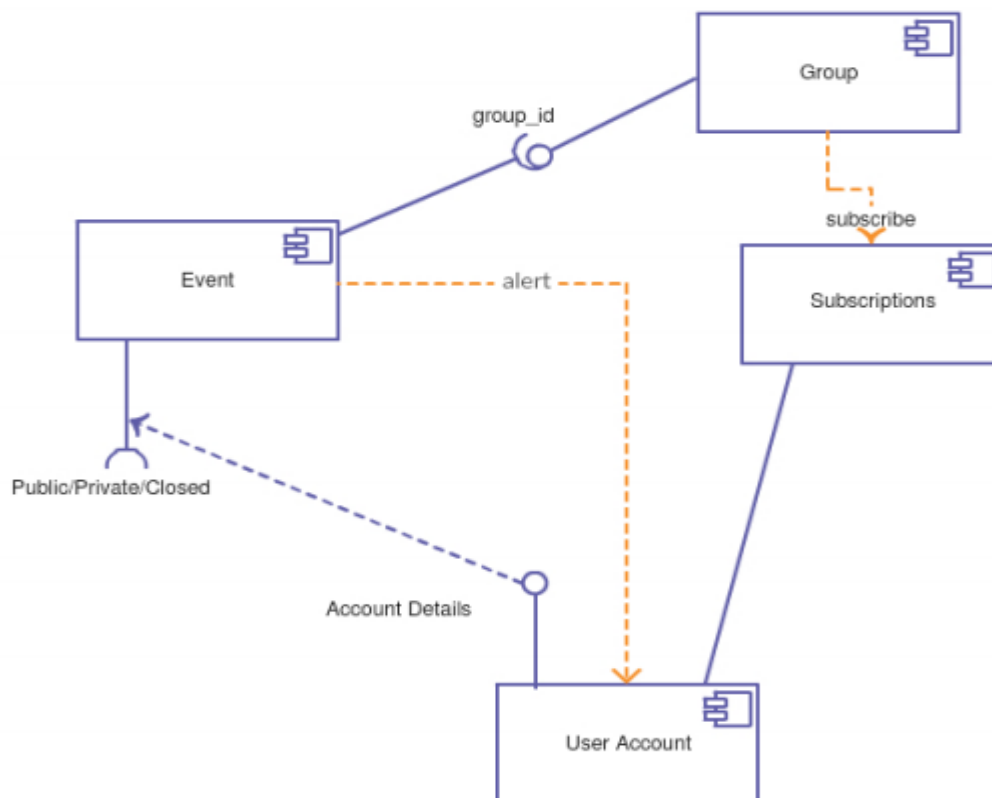


Fig 1. Architecture Diagram

3.2 User Class

The User class represents the user accounts created on the website. Every user object represents an entry in the users collection. Every user has a unique entry in this collection.

3.2.1 Processing narrative (PSPEC)

A unique record in the users collection is created when the user creates an account on the goChron website. The user object contains the following attributes:

- Name
- Email (UF Email)
- Password

3.2.2 Interface description.

Getter and setter methods are not required in the Ruby language. These methods can be activated on the desired attributes by just writing ``attr_accessor :attribute`` inside the class.

- `create(user_details : Hash)`
- `login(email: String, password : String)`
- `subscribe(group_id : String)`
- `groups() : Array`
- `events() : Array`
- `subscriptions() : Array`
- `attending() : Array`

3.2.3 Processing Detail

The objects of User class are records of the users collection which serve as a means to communicated between the rails server and the mongodb server. There are no complex processing or algorithms associated.

3.2.3.1 Class Hierarchy

User class has no parent or child classes.

3.2.3.2 Restrictions / limitations

User Class cannot create multiple instances for the same email. Every email has to be validated for uniqueness in the back-end.

3.2.3.3 Performance issues

There are no performance issues. There is only one object / record per user, and the information saved per user is also minimal.

3.2.3.4 Design constraints

There are no design constraints.

3.2.3.5 Processing detail

create(user_details : Hash)

Processing narrative (PSPEC)

When the user registers for a new account on the website, all the form details such as, name, email, password, etc. are passed in the `user_details` hash to the create() method. This creates a new record in the users collection.

Algorithmic model

IF (email already exists in collection) || (Invalid input data)

Do not create a new user.

Warn the user about the validation issues.

ELSE

Create a new user

login(email : String, password : String)

Processing narrative (PSPEC)

When the user enters the log-in credentials, the log-in form passes the user's email and password to the login() method. This looks up the email in the database and then compares the password with the hash and salt present in the record corresponding to the email. If the credentials are successfully verified the user is logged in.

Algorithmic model

IF (email not found)

User does not exist.

ELSE

IF (password does not match)

Incorrect password.

ELSE

User logs in.

subscribe(group_id : String)

Processing narrative (PSPEC)

The user can subscribe to one or more groups access all the events listed under them.

Algorithmic model

IF (User already subscribed to the group)

Do not subscribe.

ELSE

Subscribe to the group.

subscriptions() : Array

Processing narrative (PSPEC)

Gives a list of all the groups that the user has subscribed to.

Algorithmic model

GET all groups the user has subscribed to

groups() : Array

Processing narrative (PSPEC)

Gives a list of all the groups the user has created. The user is considered the administrator of these groups.

Algorithmic model

GET all groups the user has created

events() : Array

Processing narrative (PSPEC)

Gives a list of all the events the user has created.

Algorithmic model

GET all events the user has created.

attending() : Array

Processing narrative (PSPEC)

Gives a list of all the events the user is attending or attended in the past.

Algorithmic model

GET all events the user is attending.

3.3 Event Class

The Event class represents the events created on the website by the users. An event can be created either under a group or can be independent. Any event that is created independently has three attributes that ensure the level of access for website users, namely: public, closed and private. Public events are visible to all, closed events are visible to all but users need to contact the event host to get access, and private events are visible to the invited users only.

3.3.1 Processing narrative (PSPEC)

Every event object represents an entry in the events collection. An event can have several user attending, similarly several users can attend an event. But, an event can be created by one user only. The event object contains the following attributes:

- Title
- Date Time

- Description
- Location
- Link
- Access

3.3.2 Interface description.

Getter and setter methods are not required in the Ruby language. These methods can be activated on the desired attributes by just writing ``attr_accessor :attribute`` inside the class.

- `create(event_details : Hash)`
- `user() : String`
- `attendees() : Array`

3.3.3 Processing Detail

The objects of Event class are records of the events collection which serve as a means to communicated between the rails server and the mongodb server. There are no complex processing or algorithms associated.

3.3.3.1 Class Hierarchy

Event class has no parent or child classes.

3.3.3.2 Restrictions / limitations

An event can be created by one user only, but it can have many attendees. An event can belong to one group only.

3.3.3.3 Performance issues

There are no heavy performance issues. As the ``events`` collection grows too big, sharding can be a possible solution to reduce look up time.

3.3.3.4 Design constraints

If the user changes the event's access type (eg: from closed to private), several things like notifications, invitations and alerts change accordingly and must be seriously taken into account. One solution to this is, freeze the access attribute permanently once the event has been created.

If a group is deleted from the groups collection, the mapping between the group and events is permanently lost. The group admin won't have access to the events under that group anymore.

3.3.3.5 Processing detail

`create(event_details : Hash)`

Software Design Document

Processing narrative (PSPEC)

When the user creates a new event on the website, all the form details such as, title, datetime, location, link, access, etc are passed in the `event_details` hash to the create() method. This creates a new record in the events collection.

Algorithmic model

IF (Invalid input data)

Do not create a new event.

Warn the user about the validation issues.

ELSE

Create a new event.

user() : String

Processing narrative (PSPEC)

Returns the user object of the user who created this event.

Algorithmic model

GET user who created this event.

attendees() : Array

Processing narrative (PSPEC)

Gives a list of the attendees for the event.

Algorithmic model

GET all attendees for the event.

3.4 Group Class

The Group class represents the groups created on the website by the users. A group can be a department / organization / club / society, etc. The concept of group comes into play when a user posts events. User can select which group the event should be posted under.

3.4.1 Processing narrative (PSPEC)

Every group object represents an entry in the groups collection. A group can belong to any number of subscribers (users) and similarly a user can subscribe many groups. But, a group can be created by one user only. The group object contains the following attributes:

- Name
- Title
- Description

3.4.2 Interface description.

Getter and setter methods are not required in the Ruby language. These methods can be activated on the desired attributes by just writing ``attr_accessor :attribute`` inside the class.

- `create(group_details : Hash)`
- `events() : Array`
- `subscribers() : Array`
- `admin() : String`
- `subscribers() : Array`

3.4.3 Processing Detail

The objects of Group class are records of the groups collection which server as a means to communicated between the rails server and the mongodb server. There are no complex processing or algorithms associated.

3.4.3.1 Class Hierarchy

Group class has no parent or child classes.

3.4.3.2 Restrictions / limitations

A group can have only one admin, i.e a group can be created by one user only. But a group can have many subscribers.

3.4.3.3 Performance issues

There are no heavy performance issues. As the ``groups`` collection grows too big sharding can be a possible solution to reduce look up time.

3.4.3.4 Design constraints

There are no design constraints.

3.4.3.5 Processing detail

create(group_details : Hash)

Processing narrative (PSPEC)

When the user creates a new group on the website, all the form details such as, name, title, description, etc are passed in the `group_details` hash to the create() method. This creates a new record in the groups collection.

Algorithmic model

IF (Invalid input data)

Do not create a new group.

Warn the user about the validation issues.

ELSE

Create a new group

events() : Array

Processing narrative (PSPEC)

Gives a list of all events that belong to the group.

Algorithmic model

GET all events for the group

subscribers() : Array

Processing narrative (PSPEC)

Gives a list of the subscribers for the group

Algorithmic model

GET all subscribers for the group

admin() : String

Processing narrative (PSPEC)

Returns a User class object of the user who created the group

Algorithmic model

GET admin of the group

3.3 Dynamic Behavior for Components

There are 3 main components such as User, Event, Group. The main actor will be the user. For instance, the user wants to sign up for goChron for the first time. Hence, he/she needs to fill the sign up page with basic details that could create his/her account. As shown in the sequence diagrams below, the user clicks on the sign up button on the UI, the request is passed on to the server which returns the sign up form. Once, the user fills it and clicks on the submit button, the information is passed on to the server in which the authentication takes place. If invalid, the message is sent back asking the user to correct the information. If valid, the user information is successfully stored in the database and the account is created.

Similarly, the user would want to either create events or get the information related to events and many such. If the user wants to create an event, he/she will click on create event button, adds the event details in the event creation form, and the request is sent to the server. The server processes the request, sends the event related information to database. On successful addition to the database, server receives a message and thus the event created confirmation message is passed on to the user.

In addition, the user would want to either create groups or get the information related to existing groups and many such. If the user wants to create a group, he/she will click on create group button, adds the group details in the group creation form, and the request is sent to the server. The server processes the request, sends the group related information to database. On successful addition to the database, server receives a message and thus the group created confirmation message is passed on to the user.

3.3.1 Interaction Diagrams

A sequence diagram for certain use cases are presented for each component.

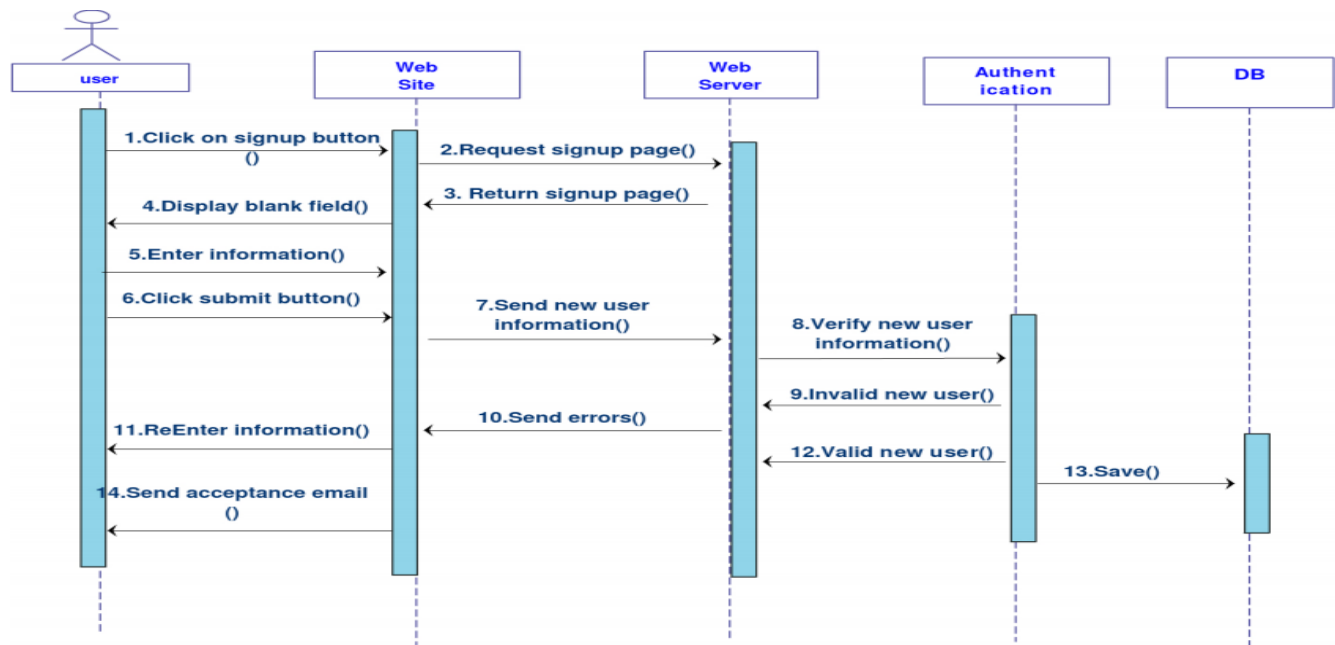


Fig 2. Sequence diagram for user Signup

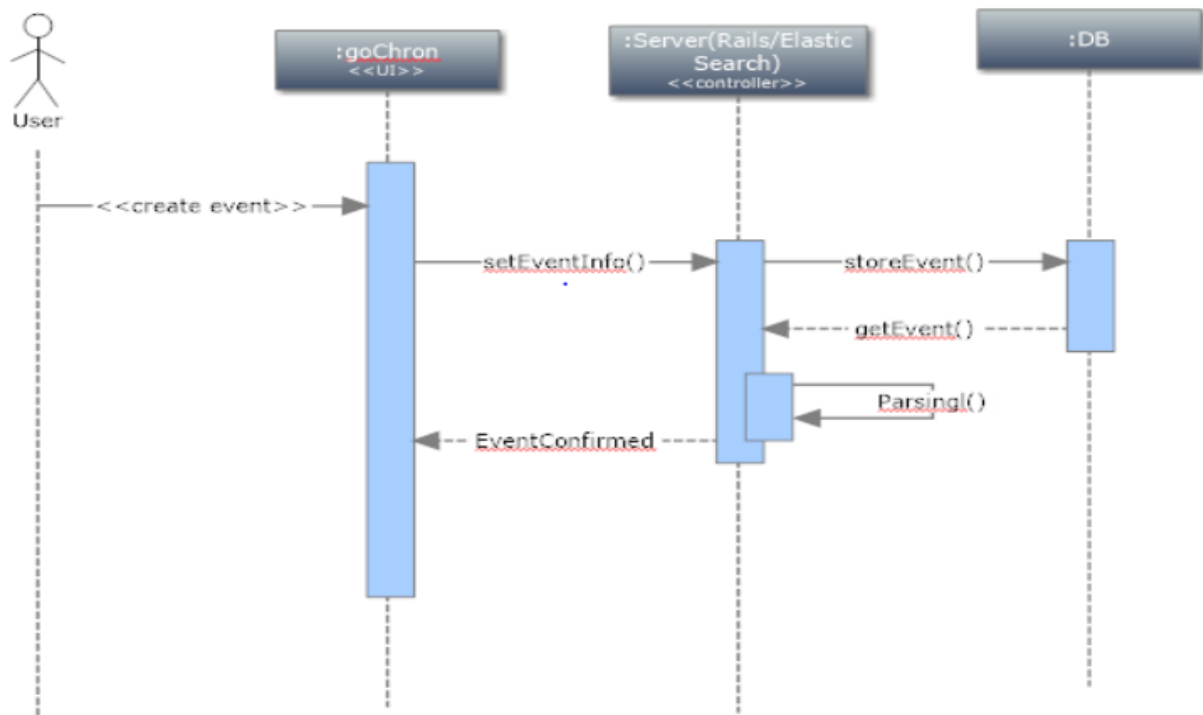


Fig 3. Sequence diagram for create event.

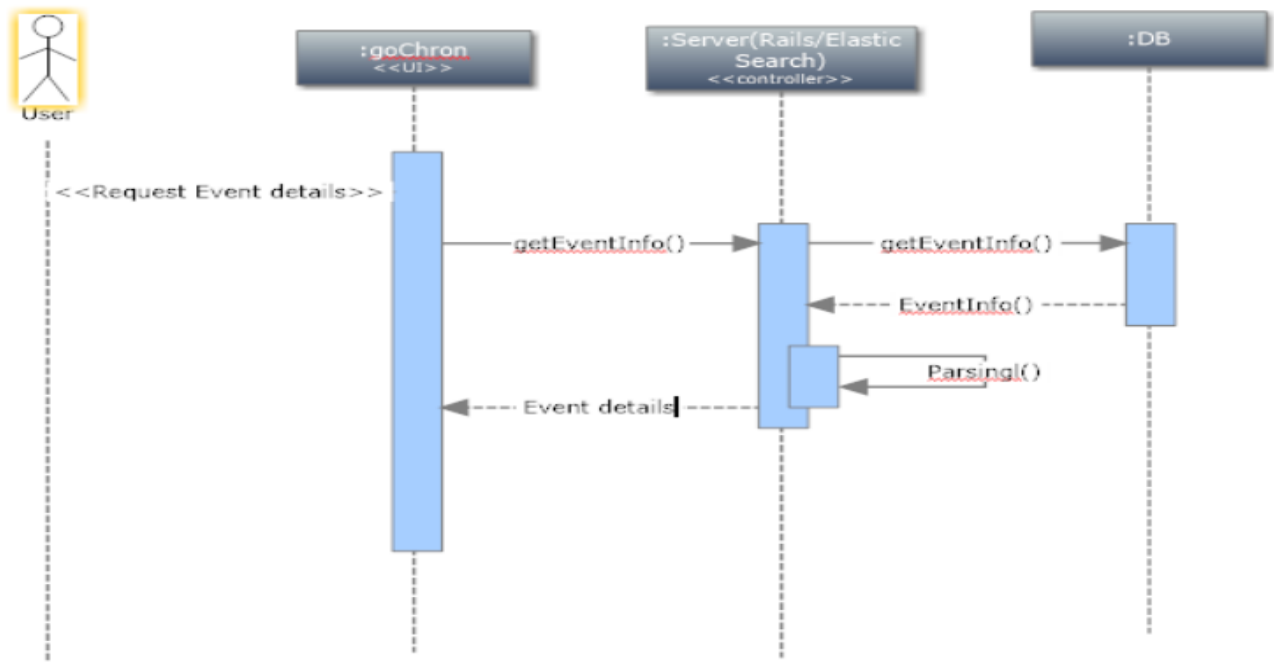


Fig 4. Sequence diagram for getting event details

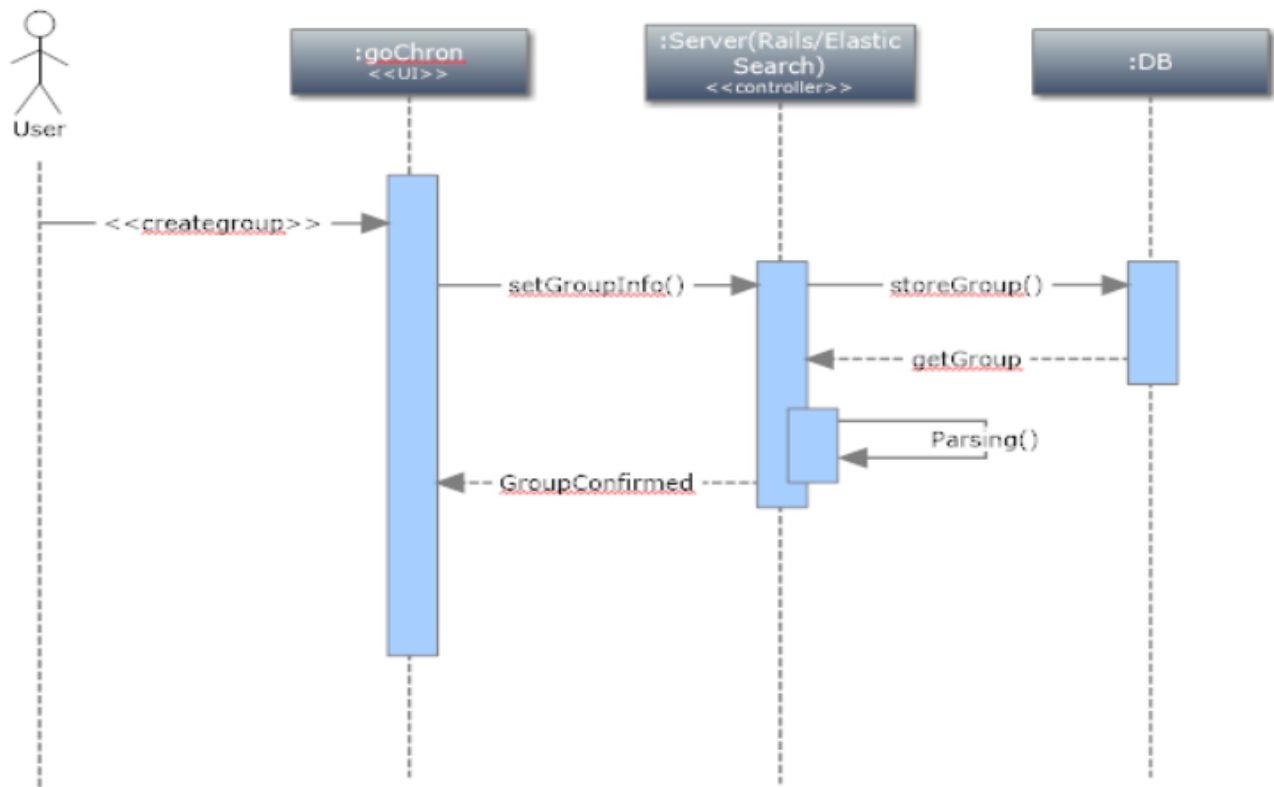


Fig 5. Sequence diagram for user creating a group

4.0 User interface design

The user interacts with the database through the web application's user interface design, the main aim of which is to make the user able to easily utilize the functionalities provided by the application without any ambiguities. The web application revolves around few pages that the user will be redirected from one to the other. They are - the "Log in/Sign up" page, "Landing" page, "Create Events/ Groups" page e.t.c.

4.1 Description of the user interface

Each page contains some basic elements like the logo, background etc., and a lot of intractable fields like the text fields, buttons to submit data e.t.c.

4.1.1 Screen images

Log in/ Sign up Page




The screenshot shows a web page titled "Log in/ Sign up Page" for "goChron". In the top left corner, there is a logo featuring a stylized alligator with "UF" above it and "goChron" below it. The main heading "goChron" is centered at the top. Below the heading is a login/signup form. The form has two input fields: "User Name" (which contains the placeholder text "Email Address" and a user icon) and "Password" (which contains the placeholder text "Password"). Below these fields are two buttons: "Sign In" and "Sign Up". At the bottom of the form, there is a link labeled "Forgot Password?".

Fig 6

- The first page seen by any user will be the above screen. If the user is a returning user, he/she will be able to log in using their credentials or also reset their password by using the 'Forgot Password' link. If it is a new user, he/she can click on the Sign Up button to get to the sign up page to register.

Sign Up Page



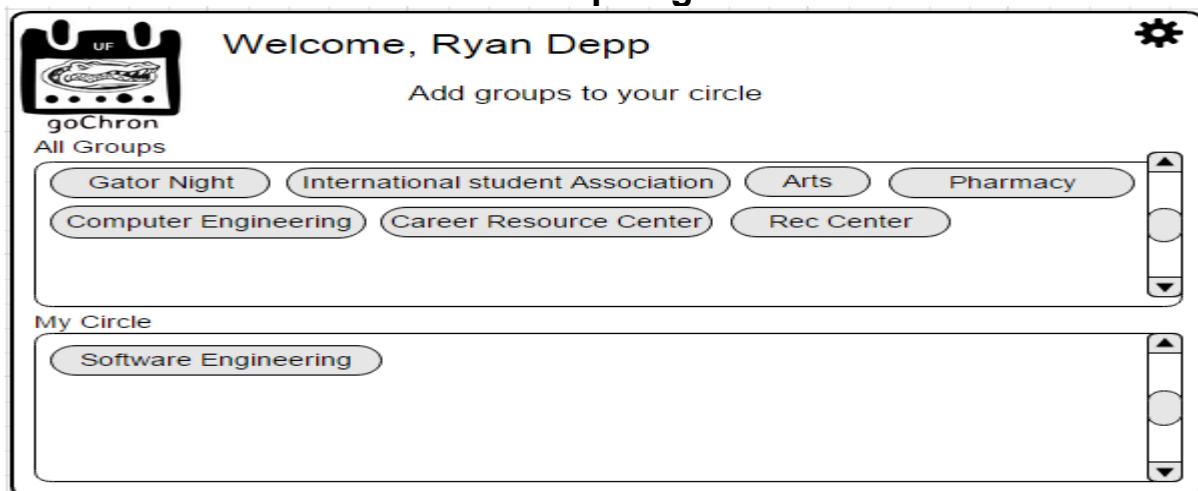
The Sign Up Page mockup features a header with the 'goChron' logo (a stylized alligator head with 'UF' above it) on the left and the 'goChron' text logo on the right. Below the logo is a registration form with the following fields and controls:

- User Name:** A text input field containing 'Ryan Depp' and a user icon.
- Email ID:** A text input field containing 'ryan.depp@ufl.edu' and an email icon.
- Password:** A text input field containing 'Password'.
- Confirm Password:** A text input field containing 'Password' and a checkmark icon.
- Buttons:** A 'Register' button with a user icon and a 'Reset' button.

Fig 7

- The sign up page allows the user to create an account by providing the required information like user name, email id, password.
- After valid information is provided, the user holds an account in the web application.

Group Page



The Group Page mockup features a header with the 'goChron' logo on the left, a 'Welcome, Ryan Depp' greeting, and a settings gear icon on the right. Below the header is a section titled 'Add groups to your circle'. This section is divided into two parts:

- All Groups:** A list of groups represented by buttons: 'Gator Night', 'International student Association', 'Arts', 'Pharmacy', 'Computer Engineering', 'Career Resource Center', and 'Rec Center'. A vertical scrollbar is on the right.
- My Circle:** A list of groups represented by buttons: 'Software Engineering'. A vertical scrollbar is on the right.

Fig 8

- After creating an account for the first time and logs into the application, the user is given a screen where he/she can add few groups of interest because it is the first time.
- This can be reached later through the settings menu.

Landing Page

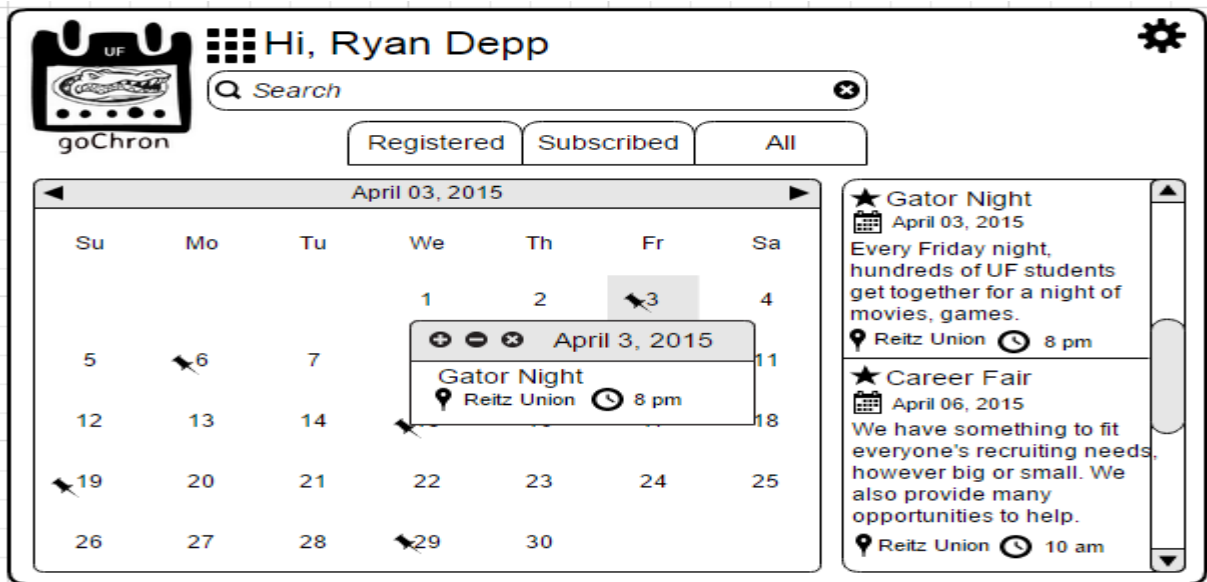


Fig 9

- This is the page that is very important to the user. This is where a lot of data is represented and shown to the user in an unambiguous way and concise way.

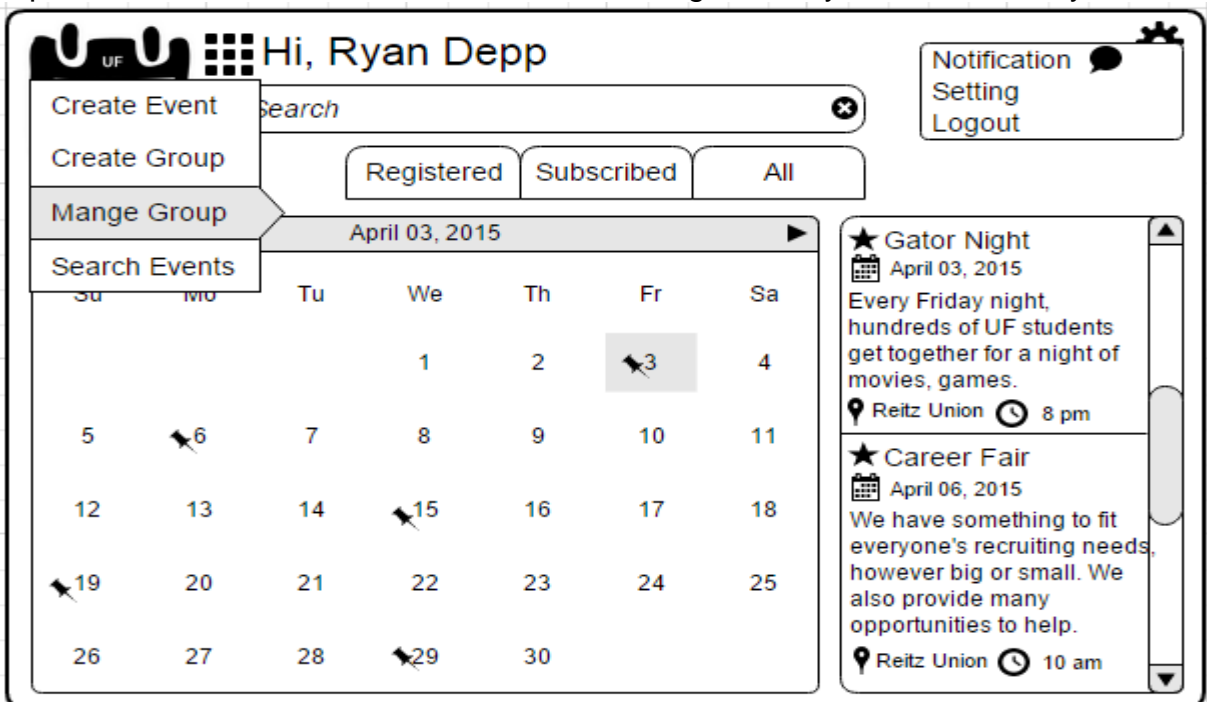


Fig 10

- The user can perform additional functions like creating groups, creating events, managing groups or events etc.,

Creating Events

- The user can create events by providing all the information required like time and date, type of the event, etc.,

The screenshot shows the 'Create an Event' form in the goChron app. The app's header includes the goChron logo, the user's name 'Hi, Ryan Depp', and a settings gear icon. The form is titled 'Create an Event' and contains the following fields: 'Event' (text input with placeholder 'Name of your event'), 'Location' (text input with placeholder 'Location'), 'Date' (calendar icon and input '10/22/2015'), 'Time' (time pickers for '12 pm' and '4 pm'), 'Type of an event' (dropdown menu with 'Private' selected), and 'Description' (text area with 'Closed' and 'Public' options). A 'Create' button is at the bottom. To the right, a 'My Events' section lists two events: 'Standardized Testing' (Sponsored by The Gainesville Sun and the Bob Graham Center for Public Service, 10/21/2015, 10 am to 11:30 am, E125, CISE) and 'Study Abroad Fair' (Sponsored by UF International Center for all department, 10/21/2015, 2 pm to 4 pm, MSL).

Fig 11

The screenshot shows the 'Create Group' form in the goChron app. The app's header includes the goChron logo, the user's name 'Hi, Ryan Depp', and a settings gear icon. The form is titled 'Create Group' and contains the following fields: 'Group Name' (text input with placeholder 'Name of your group'), 'Description' (text area), 'Group Name' (dropdown menu with 'Closed' selected), and 'Send Invite' (text area with 'Public' option). A 'Create' button is at the bottom.

Fig 12

- The user can create a group by specifying the type of the group and selecting people to send invites to etc.,



Fig 13

- The user can add events of his/her own to the calendar.

4.2 Interface design rules

1. Strive for consistency

Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.

2. Enable frequent users to use shortcuts

As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

3. Offer informative feedback.

For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

4. Design dialog to yield closure.

Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the

operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.

5. Offer simple error handling.

As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.

6. Permit easy reversal of actions

This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

7. Support internal locus of control.

Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.

8. Reduce short-term memory load.

The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of *actions*.

4.3 Components available

The web application contains a lot of components provided by HTML. They are

- Text Field
Text fields are used to get input from the user like user name and email while logging in or signing up.
- Password
Password field to get password as input from the user.
- Button
Button clicks are used to trigger events or submit information or have a functionality.

5.0 Restrictions, limitations, and constraints

The major limitation of goChron is that it is not a native mobile application. If it is designed to be one, it will be easier for him / her to access the calendar and events easily. But designing the mobile application totally depends on the time spent designing and implementing the

native web application. Once the web application is designed, a mobile app can be developed with a relative ease.

A major design constraint is to make the front end of the application compatible with any device screen. Also, since this is primarily a browser based app, smartphone users might have a UI experience only through mobile browsers unlike the native android/ios apps, but the idea of web application is to make it easy for the groups to add the events with utmost ease on a large scale.

A minor constraint is that as it is a web application it requires Internet whenever the calendar has to be accessed. This is essential as all the data regarding the events is stored in the server. The user is notified with the most recent updates whenever he logs into the application.

Another constraint imposed on the user by goChron is that they must have an email address. This is used as an unique identifier for each user. Every UF student by default has this as their ufl email and password will be the required login. Once the user provides authentication, cookies are stored and he need not from the next time he logins from his / her system.

Other constraints involve limited memory, storage space and processing power. As the traffic on the website grows, the backend architecture will have to be scaled up accordingly. For eg: Load balancers will have to be used for distributing requests to servers, database sharding and replication will have to be done for quicker access and fault tolerance, servers dedicated only for in-memory caching will be required.

6.0 Testing Issues

Different types of testing will be performed to ensure smooth functionality of the product. Starting with unit test cases, each functionality will be tested as soon as the development of that particular module is finished. Integration testing would be performed after every module is integrated.

6.1 Classes of tests

White box testing:

As a functionality is being coded, the developer also tests his functionality with junit testing as he can debug the code immediately to ensure the basic functionality of the product is working. The junits are written in such a way to ensure maximum code coverage. The developer also ensures that junits are written when a functionality is linked with another.

Black Box testing:

After integration of all the software modules, an end to end testing is performed considering different real time scenarios. Also, load testing, performance testing and security testing are performed on the final product. A user group will be picked to perform alpha testing and then website would be launched in beta version.

Functional Testing:

After completion of each module, the module will be tested by the tester extensively considering all the corner cases. The same would be repeated on integration of a new module to the existing system. A brief overview of few functional tests are as follows,

6.1.1 *Login Feature Testing:*

- a. A first time user tries to login with a valid user id. The user cannot login. The system should throw proper error message.
- b. An already registered user types an invalid username/password. The user cannot login. The system should throw proper error message.
- c. A registered user types a valid username and password. The user should be able to login and redirected to his home page.

6.1.2 *Signup Feature Testing:*

- a. The user enters an invalid email address. User should not be able to register, proper error message is thrown.

- b. The user enters a valid email address. The system has to verify the user and redirect him to login page.

6.1.3 *Creating/Removing/Editing a Group:*

- a. The user tries to create a group with an already existing group name. Group creation has to fail, with proper error message thrown
- b. The user tries to create a group with proper information. The group has to be created and displayed in the list of groups.
- c. The user tries to remove a group he created. The group has to be removed from the database.
- d. The user tries to remove a group created by a different user. The group should not be removed and proper error message must be displayed.
- e. The user tries to edit his own group with valid changes. The group details has to be updated.

6.1.4 *Creating/Removing/Editing an Event:*

- a. The user tries to create an event with incomplete information. The event should not be created.
- b. The user creates a public event, the event should be created and updated in the database and displayed in list of events.
- c. The user creates a private event and notification should be sent only to the users he added.
- d. The user should be able to edit the event he created with valid information.
- e. The user tries to remove his own event, event should be deleted.

6.1.5 *Attending an Event:*

- a. The user tries to attend a public event. The event should be added to the user calendar.
- b. The user tries to attend a closed event. The user should be put on hold and the event should be added to his calendar after the admin approval.
- c. The user receives an invite for private event. The user should be able to add the private event to his calendar.

6.2 Performance bounds

The website is built on latest technologies and the response time from server to client is expected to be quick. It is also advisable for users to have latest version of browsers for good results. Since the website is almost a single page application, with only few dynamic information changing, it should take very less time to perform different tasks on the web page. One major performance issue would be mining huge chunks of data with related key terms when the user wants to search. But by using Elastic search and adding tags in the database, we try to enhance the speed of the search. Also for a faster access of data, frequently used data is put in redis.

6.3 Identification of Critical Components

Each feature is a critical component for smooth functioning of the software. However narrowing down a few, Security and privacy of the user are given utmost priority. Since we have events categorized under private and closed, we have to make sure that people only with proper access will be able to view or attend the event. And also a user should not be able to edit/remove the data created by another user. The user calendar is private to the user and should not be accessed by other users.

7.0 Appendices

7.1 Packaging and installation issues

No packaging or installation procedures to be followed since this is a web based application. But requires the users to be able to reach the website using the assigned URL (Uniform Resource Locator) to receive the services.

7.2 Design metrics to be used

- The performance of the application can be measured by the rate at which the data requested by the user is presented.
- Scalability is measured by checking how well the application screens fit in on the screen of the device used.
- Abstractness can be known by analyzing how well the data that is required is presented concisely to the extent the users needs the data and not presenting the data directly but maybe through few more user actions the data that is least used by the user.

7.3 Supplementary information

- **Sequence Diagram**

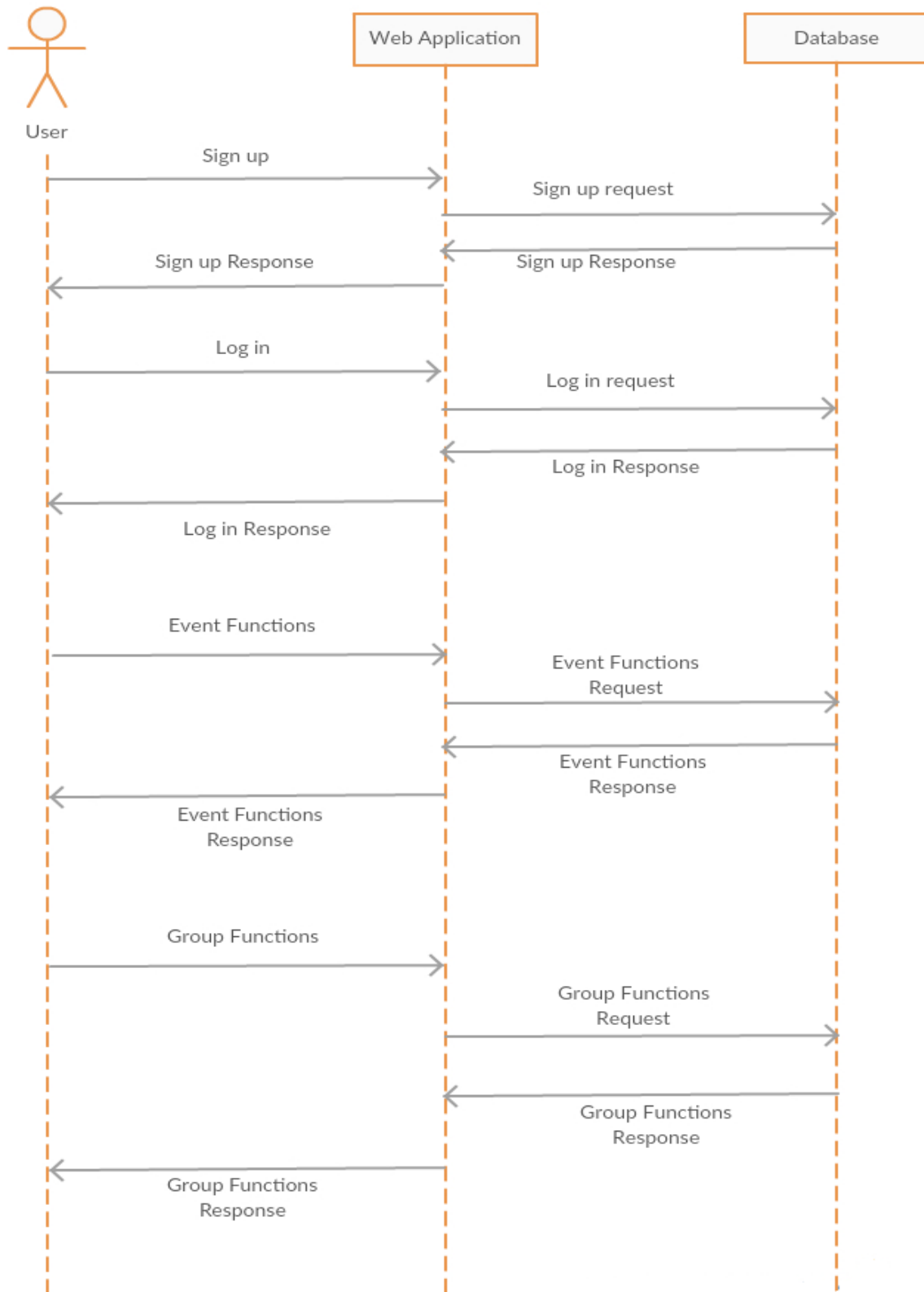


Fig 14

- Use Case Diagram

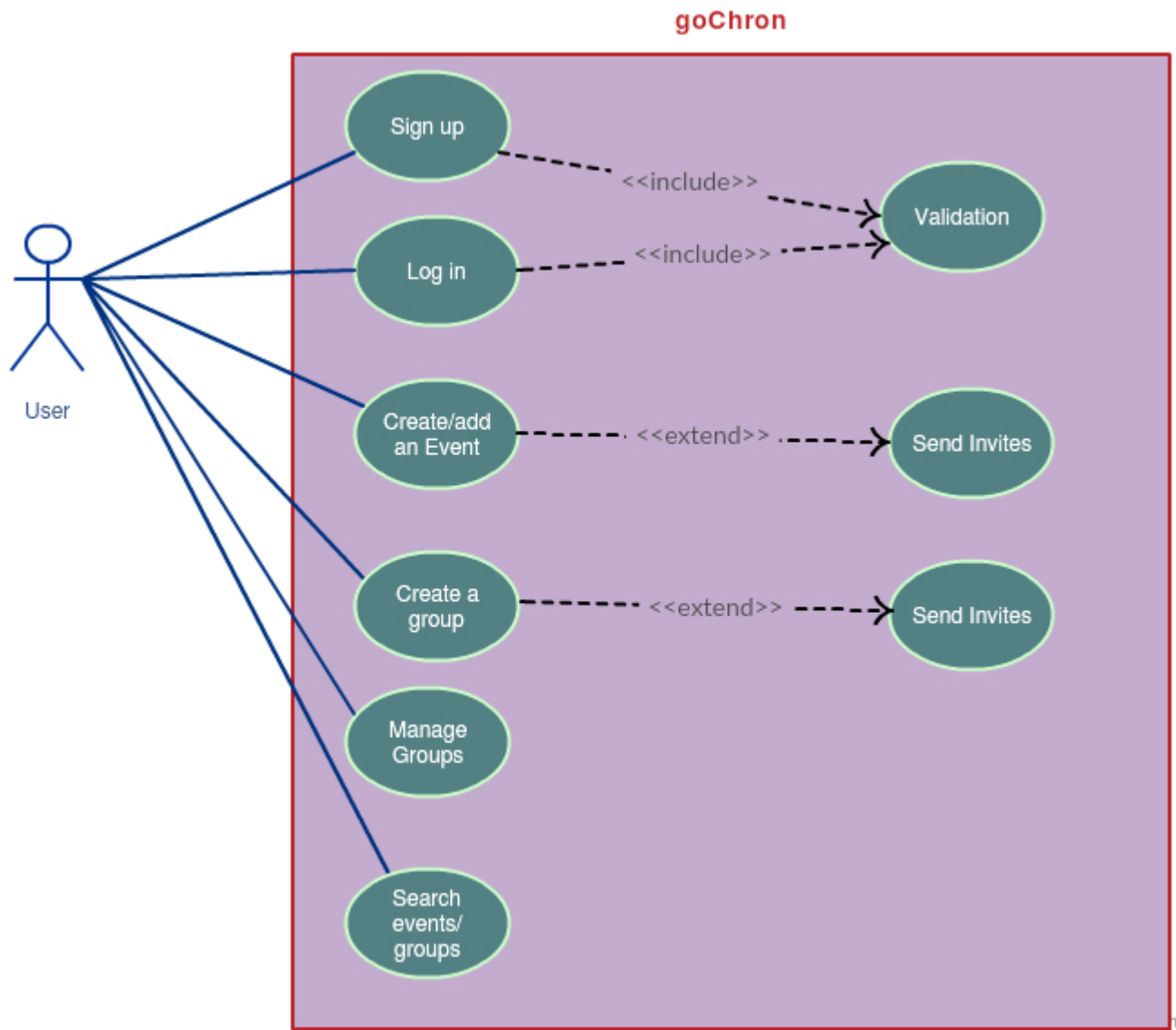


Fig 15