# Model

## Model 1: User

Default User

## Model 2: Contact

* User 1
* User 2

## Model 3: Node

* meeting id
* Timestamp
* Sender user

### Model 3.1: Remind node (inherits Node)

* Receiver user
* Message

### Model 3.2: Join node (inherits Node)

* Receiver user

### Model 3.3: Submit node (inherits Node)

* Calendar\_id

### Model 3.4: Poll node (inherits Node)

* Calendar\_id (intersected)
* Four selected datetime

### Model 3.5: State node (inherits Node)

* Date time
* State (ready or final)

### Model 3.6: Remove node (inherits node)

* Initiator
* Kicked out user

## Model 4: meeting

* meeting id
* Calendar (mutual between users)
* meeting -> List of users
* meeting -> List of nodes
* State (enum)

## Model 5: Calendar

* Owner
* meeting
* Start date
* End date
* List of events
* calendar\_id

## Model 6: Event

* Name
* Description
* Availability (enum)
* Start DateTime
* End DateTime
* Repeat & time (extra)

## Model 7: Member

* User
* Meeting
* Role

# API

Root: <http://localhost:8000/api/>

* GET: Return API document
  + Header:X
  + Param:X
  + Body:X
  + Response:X

## Accounts APP

Users: <http://localhost:8000/api/accounts/>

* GET: Return a list of all users
  + Header: Authorization(admin)
  + Param:X
  + Body:X
  + Response: id, username, email
* POST: X
* PUT: X
* DELETE: X

Register: [http://localhost:8000/api/accounts/register](http://localhost:8000/api/accounts)/

* GET: X
* POST: Create new users
  + Header: X
  + Param:X
  + Body: username, password, email
  + Response: message
* PUT: X
* DELETE: X

Profile: [http://localhost:8000/api/accounts/profile](http://localhost:8000/api/accounts)/

* GET: Return user information
  + Header: Authorization(user)
  + Param:X
  + Body:X
  + Response: id, username, email
* POST: X
* PUT: Update user information
  + Header: Authorization(user)
  + Param: X
  + Body: username, password, email
  + Response: message
* DELETE: X

Contacts: [http://localhost:8000/api/accounts/contacts/](http://localhost:8000/api/accounts)

* GET: List of all contacts of user <ID>
  + Header: Authorization(user)
  + Param:X
  + Body: X
  + Response: user1, alias1, user2, alias2
* POST: Create a new contact for user <ID>
  + Header: Authorization
  + Param:X
  + Body: user2, alias2
  + Response: user1, alias1, user2, alias2
* PUT: X
* DELETE:X

Contact ID of User ID: [http://localhost:8000/api/accounts/contacts/<contact\_id>/](http://localhost:8000/api/accounts)

* GET: Return contact <contactID> of user <userID>
  + Header:Authorization(user)
  + Param:X
  + Body:X
  + Response: id, user\_id, alias
* POST: X
* PUT: Return updated information of contact <contactID> of user <userID>
  + Header:Authorization(user)
  + Param:X
  + Body:alias2
  + Response:message
* DELETE: Delete contact <contactID> and Return an empty object
  + Header:Authorization(user)
  + Param:X
  + Body:X
  + Response:message

## Meetings APP

meetings: [http://localhost:8000/api/](http://localhost:8000/api/accounts)[meetings](https://127.0.0.1/sessions)/

* GET: Return a list of all meetings
  + Header: Authorization (Admin)
  + Param: X
  + Body: X
  + Response: List of meeting models
* POST: Create a new meeting
  + Header: Authorization (User)
  + Param: X
  + Body: name, description
  + Response: The new meeting model
* PUT: X
* DELETE: X

meeting ID: [https://127.0.0.1/api/meetings/<meeting\_id>/](https://127.0.0.1/session/)

* GET: Return the meeting <meetingID>
  + Header: Authorization (IsMember | IsAdminUser)
  + Param: X
  + Body: X
  + Response: Meeting model of specific meeting id
* POST: X
* PUT: Return an updated meeting
  + Header: Authorization (IsMember | IsAdminUser)
  + Param:X
  + Body: name, description
  + Response: meeting model got updated
* DELETE: Return an empty object and delete
  + Header: Authorization (IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response:message

Members in meeting ID: [https://127.0.0.1/api/meetings/<meetingID>/members/](https://127.0.0.1/sessions/)

* GET: Return a list of members in meeting <meetingID>
  + Header: Authorization(IsMember | IsAdminUser)
  + Param: X
  + Body: X
  + Response: List of member models
* POST: X
* PUT: X
* DELETE: X

Member ID in meeting ID: [https://127.0.0.1/api/meetings/<meetingID>/members/<member\_id>/](https://127.0.0.1/sessions/)

* GET: Return member
  + Header: Authorization(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response: Show a specific member model
* POST: Invite a new member in meetings who is a contactor
  + Header: Authorization(IsMember | IsAdminUser)
  + Param:X
  + Body: user2\_id
  + Response: newly invited a member model
* PUT: Update member information of <memID>
  + Header: Authorization(IsMember | IsAdminUser)
  + Param: X
  + Body: user2\_id
  + Response: updated member model
* DELETE: Return an empty object and delete member <memID>
  + Header: Authorization(IsMember | IsAdminUser)
  + Param: X
  + Body: X
  + Response: message

Calendars in meeting ID: [https://127.0.0.1/api/meetings/<meetingID>/members/<member\_id>/](https://127.0.0.1/sessions/)

* Authentication: Yes (Users in the meeting)
* GET: Return a list of calendars in meeting <meetingID>
  + Header: Authentication (IsMember | IsAdminUser)
  + Param:
  + Body:
  + Response: a list of calendar model
* POST: X
* PUT: X
* DELETE: X

Calendar ID in meeting ID: [https://127.0.0.1/api/meetings/<meetingID>/members/<member\_id>/Calendar/](https://127.0.0.1/sessions/)

* GET: Return calendar <calendarID> in meeting <meetingID>
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response: get a member’s calendar model
* POST: Return a newly created calendar
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response: new calendar model
* PUT: Return an updated calendar <calendarID>
  + Header:Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response: Updated calendar model
* DELETE: X

Events in Calendar: [https://127.0.0.1/api/meetings/<meeting\_id>/members/<member\_id>/calendar/events/](https://127.0.0.1/sessions/)

* Authentication: Yes (Users in the meeting)
* GET: Return a list of events of meeting <meetingID> and member <memberID>
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response:List of event model
* POST: Create a new event
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body: name, availability, start\_time, end\_time
  + Response: Newly created event model
* PUT: X
* DELETE: X

Event ID in Calendar: [https://127.0.0.1/api/meetings/<meeting\_id>/members/<member\_id>/calendar/events/event\_id](https://127.0.0.1/sessions/)

* Authentication: Yes (Users in the meeting)
* GET: Return event of <eventID> in meeting <meetingID>
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response: event model of specific event ID
* POST: X
* PUT: Return an updated event <eventID>
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body: name, availability, start\_time, end\_time
  + Response: updated event model
* DELETE: Return an empty object and delete the event ID
  + Header: Authentication(IsMember | IsAdminUser)
  + Param:X
  + Body:X
  + Response: message

Nodes in meeting ID: [https://127.0.0.1/meetings/](https://127.0.0.1/sessions/)<meetingID>/nodes/

* Authentication: Yes (Users in the meeting)
* GET: Return a list of timeline nodes of meeting <meetingID>
  + Header:
  + Param:
  + Body:
  + Response:
* POST: Create a new node
  + Header:
  + Param:
  + Body:
  + Response:
* PUT: X
* DELETE: X

Node ID in meeting ID: [https://127.0.0.1/meetings/](https://127.0.0.1/sessions/)<meetingID>/nodes/<nodesID>/

* Authentication: Yes (Users in the meeting)
* GET: Return node <nodeID> of meeting <meetingID>
  + Header:
  + Param:
  + Body:
  + Response:
* POST: X
* PUT: Return an updated node, **for example, update the poll node after a new poll**
  + Header:
  + Param:
  + Body:
  + Response:
* DELETE: X

# Appendix

## API Rules

GET /collection：返回资源对象的列表（数组）

GET /collection/resource：返回单个资源对象

POST /collection：返回新生成的资源对象

PUT /collection/resource：返回完整的资源对象

PATCH /collection/resource：返回完整的资源对象

DELETE /collection/resource：返回一个空文档

## Request Inputs

**Header:**

* metadata about the request
* HTTP Headers are NOT part of the URL
* if it's information about the request or the client, then the header is appropriate
* headers are hidden from end-users
* globally data
* restrict Attacks by detecting authorization on its header because a header can be accessed before the body is downloaded

**Param:**

* the query params are within the URL
* like this "tag=networking&order=newest"
* if it's the content of the request itself, then it's a parameter
* The product ID and requested image size are examples of "some detail" (or parameter) being supplied as part of the content of a request
* parameters can be seen by end-users (query parameters) on URL

**Body:**

* data of business logic
* important information
* unlike the body, proxy servers are allowed to modify headers
* data in specific kinds of requests
* you can pass tokens by the body as encoding & decoding in servers

## Authentication & Permission

Token generated by user login.

### AllowAny

### IsAdminUser

### IsAuthenticated

### IsMember

* Permission:
  + This user’s profile
  + This user’s contact list
  + This user’s contact
  + This user’s meeting list
  + This user’s meeting
    - This user’s meeting member list
    - This user’s meeting calendar list
      * This user’s meeting calendar’s event list
    - This user’s meeting node list

## User

Super User

username: csc309

Password: csc309yes!!

## 2 pointer version

Def find intersection (calendar1, calendar2):

assert(len calendar1, len calendar 2 > 0)

I, j = 0, 0

Intersection = []

While (i < len(c1) and j < len(c2)):

Start1, end1 = c1[i].start, c1[i].end

Start2, end2 = c2[j].start, c2[j].end

// determine whether intersect

If start1 <= end2 and start2 <= end1:

intersection.append(max(start1, start2), min(end1, end2))

If end1 < end2:

I += 1;

Else:

J += 1;

Return intersection

Def find(calendars):

Curr\_intersection = calendars[0]

For calendar in calendars[1:]

Curr\_intersection = find intersection (currintersection, calendar)

If curr\_intersection.length == 0:

Return []

Return current\_intersecion