Design Document

Eric Aguilera, Justin Bonner, Carson Harmon, Dan Zheng

1 Purpose

College courses are difficult and many students choose to form study groups. However, forming and managing a study group can be a hassle. Students have to juggle multiple applications, including messaging apps for communication and calendar/scheduling apps for planning meetings. There exists no centralized application designed specifically for student group study. Our product combines the best features from messaging and scheduling applications to provide the definitive group studying application for Purdue students.

Functional Requirements

Note: any user can act as a meeting creator, meeting participant, group creator, or group member. The latter types are listed separately from user' only to categorize user stories clearly.

As a user, I would like to:

1. create a Meetchu account using a valid email and password.
2. recover my password using my email.
3. add profile information (name, photo, major, phone number, short bio) to my account.
4. update my account information at any time.
5. search for courses by title and number.
6. add a course to my account.
7. view people in my courses.
8. view a list of my courses.
9. create and join study groups with people in my classes.
10. send personal messages to any other user.
11. view message history with other users.
12. see users who are online (if time allows)
13. link my account with and authenticate using Google and Facebook (if time allows).
14. view my class schedule (if time allows).
15. search for people in my classes by name or email (if time allows).
16. view the schedules of members in my study groups (if time allows).

As a meeting creator, I would like to:

1. create a one-time meeting for a study group.
2. create a personal one-time meeting with selected members.
3. suggest possible dates and times for a meeting.
4. add a title, purpose, and location to a meeting.
5. add or remove people from a meeting and notify them.
6. cancel a meeting and notify participants.
7. finalize the date and time for a meeting and notify participants.
8. view participants who have RSVP'd to a meeting.
9. add an address to a meeting and view on Google Maps (if time allows).
10. view meeting attendance (if time allows).
11. create a recurring meeting for a study group (if time allows).

As a meeting participant, I would like to:

1. view my meetings.
2. indicate my preferred times for a meeting.
3. RSVP to a meeting after its date and time have been finalized.
4. set time preferences for group meetings.
5. receive email notifications about upcoming meetings.

As a group creator, I would like to:

1. be able to delete the group and notify members.

As a group member, I would like to:

1. be able to leave the group.
2. invite people to the group.
3. send messages to the group.
4. view group messages.

Non-functional Requirements

- Performance

Our web application will prioritize performance in order to maintain a responsive user interface. Performance is important for many features in our application. Database queries about users, groups, meetings, and messages should be processed quickly so that all requests resolve as fast as possible. For example, users should be notified quickly upon any changes to a particular meeting so that everyone can be on the same page. We will use MongoDB to store all persistent data and design intelligent schemas to minimize performance impact. Finally, we will ensure that the proper data structures are used to handle our backend computations so that applications performance is optimal.

- Security

Security is vital for Meetchu as the app will store important personal information, such as user emails, passwords, personal information, and messages. We will implement the following measures to ensure the utmost security: storing hashed passwords, preventing cross site scripting, sanitizing and validating user input.

- Ease of use

Meetchu will provide an intuitive way to schedule group meetings. We want to ensure that users have a very streamlined experience with our application by making the interface as user-friendly as possible. For example, users will be able to easily click and drag to select multiple preferred times for a meeting, rather than clicking each time individually. By providing this user-friendly interface, our application will be more likely to be adopted by our target audience.

2 Design Outline

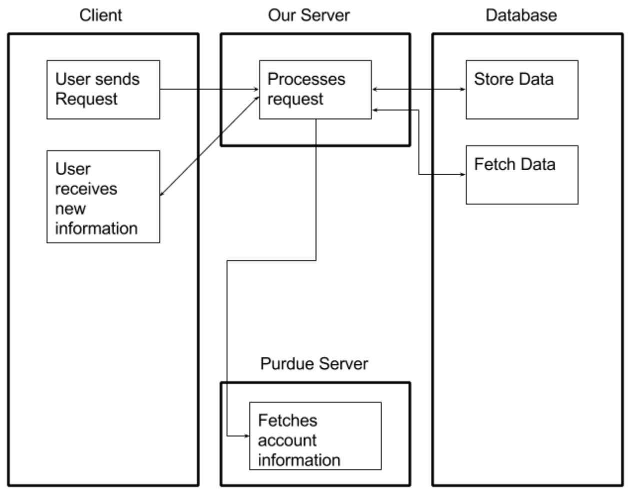


Figure 1: High level overview diagram

Our application will use a client-server architecture: a user uses a client to send requests to our server, which may send queries to or fetch data from a database. The server then returns a response, which is parsed by the client.

3 Design Issues

Architecture

Option 1: **Client-Server Architecture**
Option 2: Unified Architecture

Client-server architecture fits our needs because we need a central point to control the meeting schedule for many users. The client-server architecture also makes our code base more modular and allows for front-end and back-end capabilities. This architecture also enables us to take advantage of existing libraries that will help us design our software as a web application. A unified architecture is less modular, making it more difficult for us to collaborate. The code for a unified architecture application may also become messary

Back-end framework

Option 1: **Node.js**
Option 2: Flask

Node.js is an open-source Javascript platform designed for creating fast network applications. It is event-driven and has a non-blocking I/O model, great for handling a large number of requests. Since Node.js uses JavaScript, asynchronous operations and manipulation of JSON is easy and efficient. Node.js is a mature framework and has an enthusiastic community in addition to a huge number of open-source libraries.

Flask is used for quick prototyping of small projects: it is much more lightweight and has much fewer packages. As a microframework, it is designed for smaller projects with less complexity. For a feature-rich application like ours, Flask does not provide the same features or extensibility that comes with Node.js.

Front-end framework

Option 1: AngularJS
Option 2: **None (Pug templating engine for controller logic)**

One of the most popular solutions for a front-end framework is AngularJS. By using AngularJS we would be committed to developing a model-view-controller (MVC) architecture. However, the major downside of AngularJS is its learning curve. We decided that it would take too long for our team to be properly acquainted with such a complex framework. Due to time constraints, we will initially avoid using a framework and opt for a less restricted way of implementing our front-end. In order to create some functionality that mimics a front-end framework, we will utilize the templating engine Pug. Using Pug, we will be able to quickly add conditional logic to our templates.

Database software

Option 1: MongoDB
Option 2: **MySQL**

MongoDB is one of the most popular databases used in conjunction with Node.js. MongoDB stores objects in a JSON-like format, interfacing nicely with JSON queries sent from our Node.js server. This allows us to use a uniform language throughout our technology stack and manipulate database objects easily. However, as MongoDB is a document database, it is not suited for storing relationships between different objects, which is critical to our application.

The main selling point for MySQL, in terms of our project, is that it provides a relational-database for us to work with. Since our project has complex relationships (users, schedules, groups, etc.), it makes sense to minimize the number of database queries and simplify our application overall by using a relational database. We plan on using MySQL since it provides powerful relational queries that our application needs.

UX design

Option 1: Adaptive
Option 2: **Responsive**

Responsive design and adaptive design both attempt to optimize the user experience. Responsive design is meant to be flexible: an example is a website that resizes and adjusts its UI for mobile devices. An adaptive design detects the settings of the device and then provides the appropriate layout. While both choices are feasible and provide a good user experiene, we have chosen to use a responsive design. A responsive design is effective and is easier to develop. Adaptive design is resource heavy and more difficult to implement.

Notification system

Option 1: Text
Option 2: **Email**

Users can configure whether they want to receive notifications through email or through the messaging section of our application. There are several issues with text notifications: it requires additional user information, may potentially cost additional fees, and is generally less desirable than email notifications. Our notifications will be disabled by default to avoid undesirable spam: users who want to opt-in may do so after registering.

4 Design Details

Class Diagram

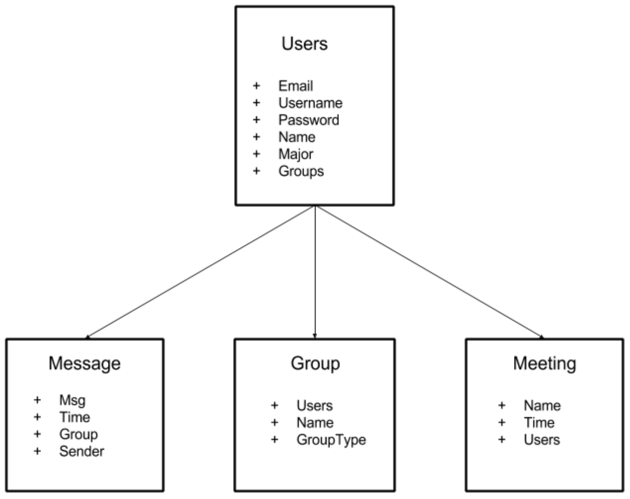


Figure 2: Class Diagram

The User class represents users of our application. Users can form groups, send chat messages, and set meeting times. A User objects contains several fields: email, username, password, name, and major. The Group class represents a group of users. A Group object contains a group name, a collection of users, and a group type, which can be either be `group` (3 or more people) or a `private-message` (2 people). The Message class represents a chat message: it contains a message string, a timestamp, a group id, and a sender id.

Account Creation and User Login

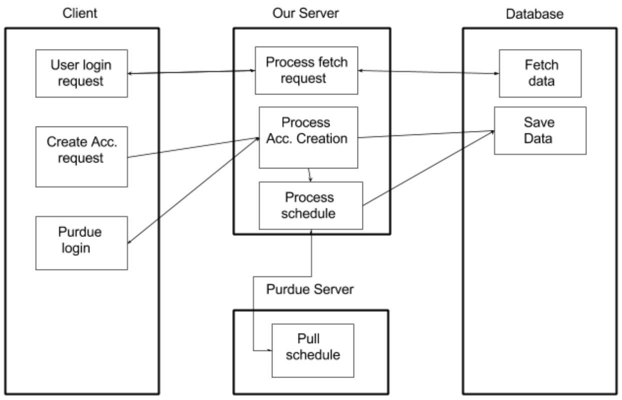


Figure 3: Account Creation and User Login

The user submits an account creation request to the server. The server then validates and processes the request, creating a user object and storing all relevant user information such as email, username, and a hashed password. The server then prompts the user for their Purdue Career account information if they would like to pull their schedule from the Purdue server. If accepted, the server then pulls the user's schedule from Purdue, and stores it in the database.

Token Authentication Process

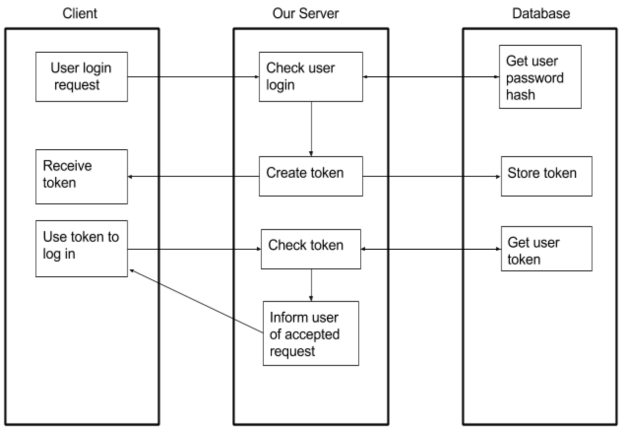


Figure 4: Authentication Process

The token authentication system starts when a user logs in with their credentials. The server then authenticates the users username and hashed password using information stored in the database. If the authentication succeeds, a session token will be stored in the database and sent to the user. This session token allows users to stay authenticated for a preset amount of time (for example, 7 days). While a user's session token remains valid and they have not logged out, they remain logged in and can perform any account-related actions.

Database Schema

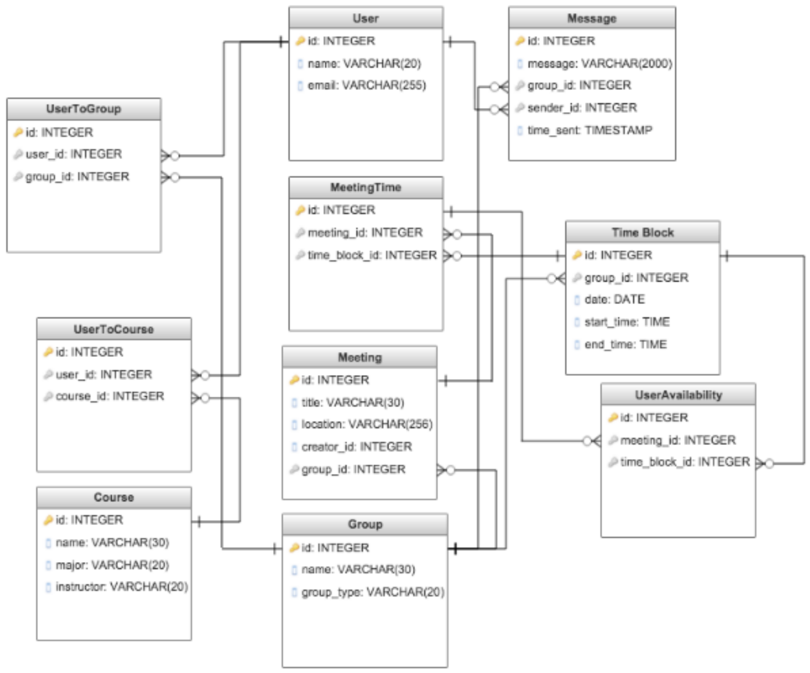


Figure 5: Database Schema

Our database schema for MySQL has 10 separate tables. Some of these tables contain foreign keys used for relational queries. Our database design utilizes One-To-Many and Many-To-Many relationships to store associations between `users↔groups`, `users↔courses`, `time blocks↔users`, and `time blocks↔meetings`.

Each table has a primary key: the `id` field. The `id` field is an auto-incrementing integer which represents a unique id for each row in a table.

The User table represents the User class: it stores properties such as users' name, email, major and any other user information and settings.

The Group table represents the Group class: it stores groups' names and their group type. The group type will be stored as a varchar in order to preserve readability in our database. There are two group types: `group` and `private-message`. In order to keep track of which groups users are in, we will create the UserToGroup table which contains foreign keys for `user-id` and `group-id`.

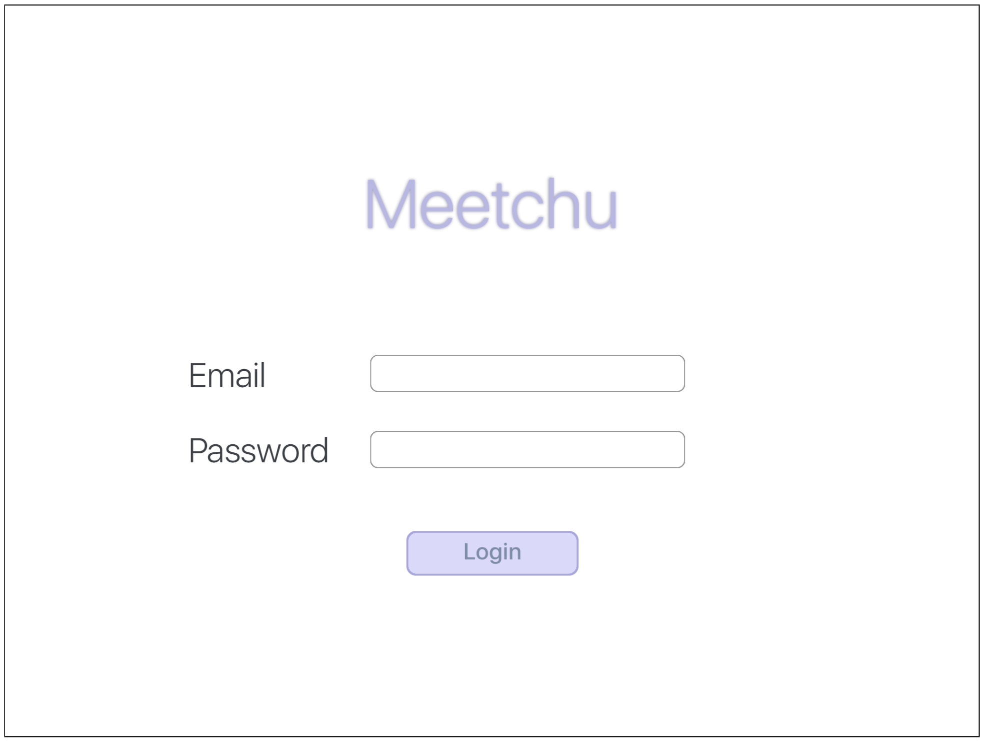
The Message table stores messages for all types of groups, it will include fields to keep track of the sender, the recipient group, the message text and the message timestamp.

The TimeBlock table is used to keep track of time ranges for possible times to hold a meeting. In order to allow a mapping between time blocks and users, we will create the UserAvailabilty table with foreign keys for `meeting-id` and `time-block-id`. Similarly, in order to mark which times are available for meetings we will have the MeetingTime table with foreign keys for `meeting-id` and `time-block-id`.

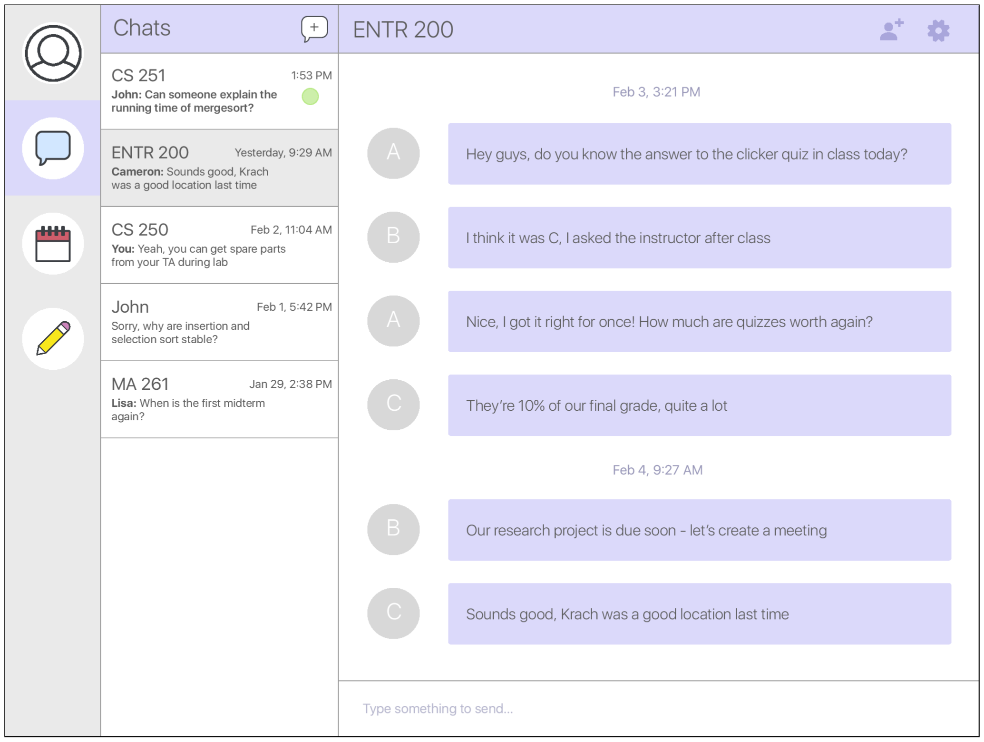
The Course table stores the list of courses that students are enrolled in, keeping track of specific properties such as the course department, instructor, and course title.

Finally, the UserToCourse table keeps track of the courses that users are enrolled in. This has foreign keys for `user-id` and `course-id`.

UI Mockups



Desktop login page.



Chats

CS 251 1:53 PM
John: Can someone explain the running time of mergesort?

ENTR 200 Yesterday, 9:29 AM
Cameron: Sounds good, Krach was a good location last time

CS 250 Feb 2, 11:04 AM
You: Yeah, you can get spare parts from your TA during lab

John Feb 1, 5:42 PM
Sorry, why are insertion and selection sort stable?

MA 261 Jan 29, 2:38 PM
Lisa: When is the first midterm again?

ENTR 200

Feb 3, 3:21 PM

A: Hey guys, do you know the answer to the clicker quiz in class today?

B: I think it was C, I asked the instructor after class

A: Nice, I got it right for once! How much are quizzes worth again?

C: They're 10% of our final grade, quite a lot

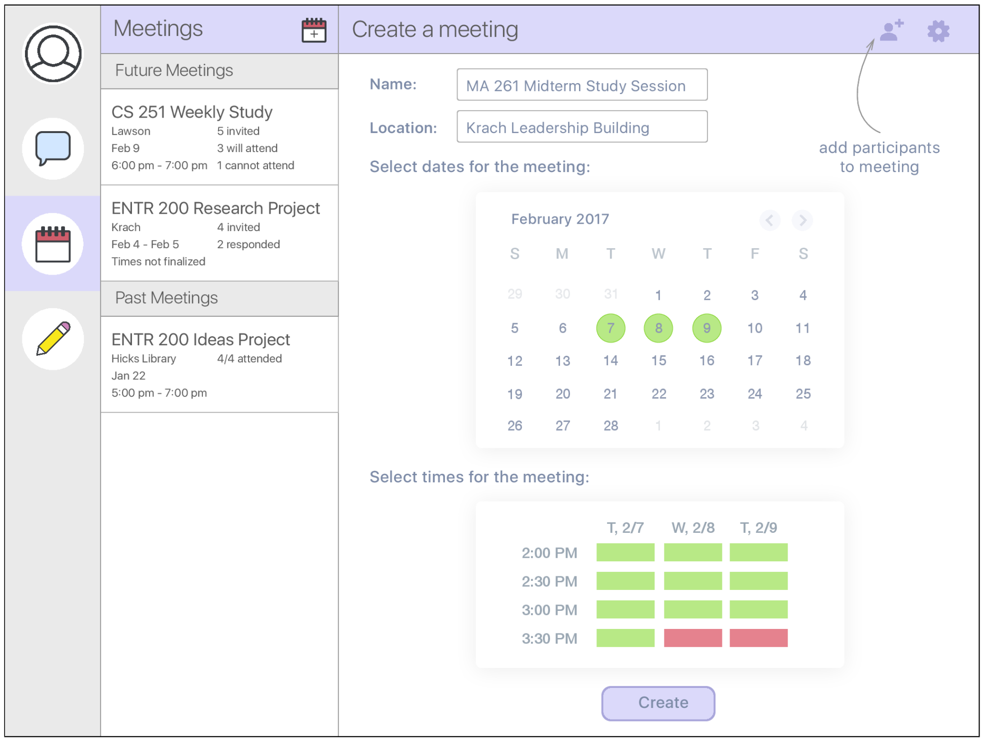
Feb 4, 9:27 AM

B: Our research project is due soon - let's create a meeting

C: Sounds good, Krach was a good location last time

Type something to send...

Messaging page. It displays group and individual chats. Users can create new chats, add people to existing chats, and send messages.



Meetings

Future Meetings

CS 251 Weekly Study
Lawson 5 invited
Feb 9 3 will attend
6:00 pm - 7:00 pm 1 cannot attend

ENTR 200 Research Project
Krach 4 invited
Feb 4 - Feb 5 2 responded
Times not finalized

Past Meetings

ENTR 200 Ideas Project
Hicks Library 4/4 attended
Jan 22
5:00 pm - 7:00 pm

Create a meeting

Name: MA 261 Midterm Study Session

Location: Krach Leadership Building

add participants to meeting

Select dates for the meeting:

February 2017

S	M	T	W	T	F	S
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	1	2	3	4

Select times for the meeting:

	T, 2/7	W, 2/8	T, 2/9
2:00 PM	🟩	🟩	🟩
2:30 PM	🟩	🟩	🟩
3:00 PM	🟩	🟩	🟩
3:30 PM	🟩	🟥	🟥

Create

Meetings page. Users can create meetings, specify a name and location, propose meeting times, and add participates.