**Coodle Design Doc**
**Milestone 1 for CS 3110 Final Project**

Wenyuan Ma (wm263)
Qihang Yan (qy33)
Tianyi Zhang (tz58)
Shiran Chen (sc2289)

- **System Description**
  - One-liner: Coodle = Calendar + Doodle.
  - Vision: Coodle is a collaborative scheduler that improves group time management. For any event, Group members submit their available time slots to facilitate the event administrator's decision of meeting time. Coodle offers a reasonable interface to display selected time slots from different members. Moreover, for group use, Coodle provides support for different admin rights and visibility option; for personal use, Coodle can also function as a calendar to display personal agenda concisely.
  - Feature list:
    - Event scheduler: For a given event, all members in the group can view other's selected time slots. This feature is further modified by visibility option. For example, event administrator can set the time slots selection to be anonymous when creating the event.
    - Group Time management: Administrator can finalize the time decision based on member's availabilities. An event is understood as open if the final time has not been decided. An event is understood as closed if the administrator(s) have finalized the time choice. Each members in the group would see different pages for events with different status, for example, when an event is open, the users will see the time slots table, as exemplified below in Demo Section. And when an event is closed, the user will see the page with the finalized meeting time.
    - Personal Schedule: For a given user, one can access to all the events that she created or was invited as an member. For open events, the calendar would indicate the time slot that this user chose or remind the user to submit time choice. For closed events, the user will be able to see the list of finalized events' schedule in "My Schedule".
    - Admin management: Within a group, there is a hierarchy of admin rights ranging from administrator, to manager, to member. The admin rights can be updated, transferred, or removed. The admin right is created when 1) a user created a new event; she became the administrator 2) a user is invited to be manager or member of an event 3) the admin right of a user is changed after the user has been invited into the event
    - Account system: The Coodle program allows user to register as a new account or use old account to keep track of her data. Users must register with unique username otherwise, an error message will be displayed.

- **System Design**
  - **Overview**

    We implemented a server-client architecture for our web application and we use a sql database on the server side to handle data. We will first explain the server-client architecture, then illustrate our data (their representation and our treatment of the data), and finally we will break down each page to fully disclose the functionality of our application.

    A word before the details: there are unfortunately some circular dependency in our illustration. For example, our description of the design of the client-server architecture refers to several specific pages in the application. That being said, we hope you may go back and forth at some moment to have a better picture of our design.
  - **Client-Server Architecture**

    Given the design of Eliom (the Web framework we used), the server side and the client side are tightly integrated. Although, many functionality can be realized on both side, our current application is really a server-heavy one. This choice is based on the consideration that as much user-friendly as possible, without assuming the capability of their devices and save the computational pressure on the server side. Even so, we also put huge effort on the client side to try to offer cooler interactions between the application and the user.

    An usual observation might be made about our project: the technical difficulty is not uniform across the project. For example, there are some redirections that may appear as cumbersome to a modern web developer and there are also more dynamical reconstruction of a single solid page which offers better user experience. This actually represents our learning curve. None of the group members have any experience in web development before, not to mention in ocaml where the support for web is relatively limited. We build up abilities along the way and we hope that this may be taken into the evaluation of our project.

    Now we naturally break our discussion into the server side and the client side, with each section breaking into subsections which are about more specific functionality.
    - **Server**

    - Give client data
      - GET Request

        The client side retrieve data from the server mostly through GET Request.

        1. For the most obvious example, we use Eliom_content.Html module to ensemble html components on the server side (This is implemented by pure ocaml, although the js_of_ocaml would compile it to javascript).
        2. Moreover, we use ocaml features to handle GET request parameters and change the web page content (HTML or

Javascripts) accordingly. This greatly reduced the number of "web services" we need to create. For example, all event pages are actually created by the same "web service" which takes a GET request. We match the GET request parameter in ocaml to tasks ranging from displaying different eventname to rejecting page visit.

3. We use ocaml to build highly "compositional" page content. Instead of hand-coding a great chunk of html code, we implement application logic in each tiny part of the object. This design is made possible by modular programming (we put reusable forms and other html content in a module Widget and use function of this module to generate the actual html object) and ocaml elegant let expression.

- Server function
  Eliom supports

■ Collect client side information.

- POST Request

  Most simple communication from client side to server side is implemented through POST Request. This including login and signup. Although it is also possible to implement by GET Request, sending data by parameter, the data shows up in the url. For safety reason, we prefer the POST request.

  By default, the POST Request is usually used with a fallback option so that the user would go somewhere after the request. However, in some cases we view this as too rigid and limited. For example, in the signup page we want to user to proceed to main page if sign up successfully and stay in the same page while being prompted by a new message. Therefore, we use ocaml to handle the result and store temporary data and use Eliom_registration.Any module to implement more complex redirection logic.

- Server Function

  Eliom offers a more advanced data collecting solution, which is server function. Instead of having the user sending POST/GET request explicitly (the page "blinks"), the client side script would "call" a server-side function. It is not as simple as it may sound like since we cannot directly call a server side function. Instead, one has to convert the function argument into json and send it to the server. Most of the ground work is done by the Eliom framework; however, this is still one of the most tricky part given the weak type-inference system on the client side. A example usage of the server function is when the user submitting the timeslots. After the user submits the timeslots, the page stays almost the same except a new message indicating the success of submission. There is no reload or redirection. This would relieve many pressure from the

server side since it would have to handle less requests. Also, this would increase user experience (less reloading time, smoother transition). The server function makes room for scalability.

- Store data:
    - Store information in database

      After the data from client side is collected, we want to store some of them in the database. Handling many clients' data at the same time require us to solve the concurrency problem. This is done through the Lwt package which is also a part of eliom.
    - Store temporary data.

      Keep track of mutable information of each session and process. Certain data are really specific to a user behaviour at a particular moment. Storing this kind of data into database is expensive and cumbersome. We exploit Eliom_reference and Eliom_cache module to implement an efficient and concise solution to handle the temporary data. We now discuss some examples.

      1. Connected user. Which user is currently using application is really a question specific to a web session. Therefore, after the server verify the login information (username and password), we store the id of the connected user in a Eliom_reference type within a session scope. Note that the user use POST request to submit the login information and after the POST request, the user would "fallback" to the main page. The server now construct the main page with respect to the session reference and return the "event page" to the user. It should be highlighted that although the server is returning a different pages, the URL stays the same. This design introduces less redirection and also prevents a connected user to login again (which usually leads to undesired consequence). This i s possible only through the powerful Eliom_reference module.
      2. Timeslot cache. User's activities on the client side can usually lead to discrepancy between server data and client data. For example, after a user selects some timeslots, the timeslots information on the client side would be different from the server side. One solution to this is that we can have the client communicates with the server side "frequently": every time the user select a timeslot, we will update the server. However, this solution would pose a lot of pressures on the server and therefore leads to worse performance. Our solution is to create a cache of data on the client side and only sync the cache with server side when user clicks "submit". This saves lots of unnecessary server-client interactions.
- **Client**

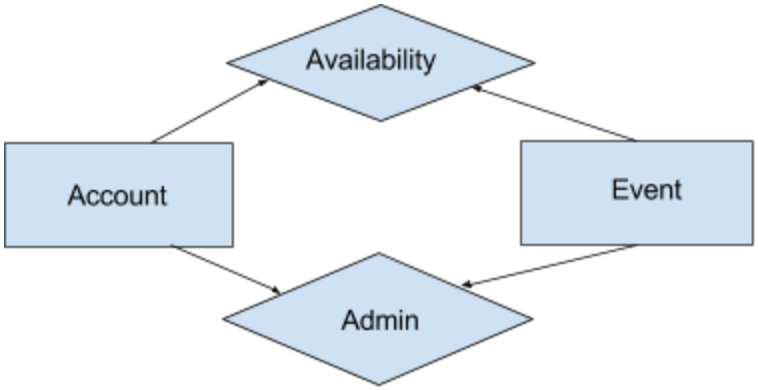
  This client side compared to the sever side is much lighter. A heavy part of the client side code it about the server-client communication, which we

have included in the server side section. Therefore, we want to highlight only this single feature.

- Dynamical Display
  We exploit the client side dynamical feature, especially handlers in Javascript. For example, in the event page, if the user hover over a selected timeslot, it would display the participants in the timeslot. This is made possible by dynamically loading html components into the display area by a javascript function (more specifically, an ocaml function using js_of_ocaml features). This is acutally a very practical feature that facilitates user selection of the time choice.

- **Database:**

There are two entity tables in our database design, Account and Event. Account stores all information related to a certain account and Event stores all information related to a certain event. Based on the two entity tables, there are two more relation tables, Availability and Admin. Availability stores all users' available time slots and Admin stores users' admin right on an event. Their relations are shown in the diagram below.




1. Account: Each Account entry carries two pieces of information:
   - *aid,* the username of the account. Every name should be unique in the database.
   - *pwd*, the password of the account. It should be nonempty.
2. Event: Each Event entry carries a record that contains
   - *eid,* the unique id (or name) of the event
   - *description*, the text that describes the contents of the event
   - *status*, either undecided or finalized. 'Undecided' means the date of the event has not been finalized; 'finalized' means the date has been finalized
   - *start_time,* the earliest date when the event can take place
   - *final_day, final_time,* the finalized date and time of the event

- *anonymous*, a boolean value indicating whether the event participation is anonymous or not. If it is true, all event participants' names should be displayed as 'Anonymous'; otherwise their names are explicitly displayed.

3. Admin: Each Admin entry represents a user's admin right on an event. It contains
    - *aid*, the username, which references the foreign key *aid* in the Account table
    - *eid*, the event id, which references the foreign key *eid* in the Event table
    - *admin_right*, either Admin, Manager or User.
4. Availability: Each Availability entry represents a time slot of a user on a certain event. It contains
    - *aid*, the username, which references the foreign key *aid* in the Account table
    - *date, time*, the date and time of the time slot
    - *eid*, the event id, which references the foreign key *eid* in the Event table

    - My Events: the events that are created by the account (the user of the account is an admin)
    - Invited Events: the events in which the user of the account is invited to edit but has not selected their available time slot yet
    - Upcoming Events: the events in which the user of the account is invited to edit and has selected their available time slot

- **Client**

  Login Page

  1. Display the username text field and the password text field in which the user can input their account information.
  2. If the user does not have a coodle account, they can click on the new user button to be directed to the signup page.
  3. If the user already has an account, they can click on the login button after entering their username and password in the textfield. Once the button is clicked, if the input username and password are valid information of a certain user, the login page jumps to the event page of that user. Otherwise, pop a warning message that informs the user about the incorrect username/password combination.

  Sign Up Page

  1. Display the username text field and the password text field in which the user can create their account information.
  2. If the user clicks on the create button, they will be directed to the login page if and only if the username and password they entered in the text field are valid username/password information.
  3. Username may not be empty, password may not be empty.

My Event Page

The event page contains three lists of clickable events.

1. My events: display all events that are created by the user. If user clicks on one event, the program will go to finalize page in which the user can finalize the event time.
2. Manager events: display all events that the user is a manager for.
3. User  events: display all events that the user is in .
4. Create Events: If user would like to create a new event, the user can do so by visiting the create event page

My Schedule Page

1. The user is able to see all the finalized events in this page. All events are sorted in chronological orders, and only the ones that are upcoming are shown in the list.
2. There is a link at the bottom of the page, “Past Events” that directs the user to the page including all the past due events.

Create Event Page

1. The user needs to enter an event name, choose a starting time and date, and select if the user wants the time slot selection of its invited users to be anonymous. The creator is also welcomed to include a short description of the event.
    a. Event name may not be empty

Calendar Page

Display the current calendar of a certain event for a user.

1. Display a table whose columns are 7 dates of next week and rows are 14 hours from the starting time by default.
2. Each box of the table represents a time slot with the time it represents as the button text label.
3. The user can click on any available time slot to select and that time slot will change color. Additionally, the page will display a list of people who have also selected that time slot. If the user has already selected available timeslots, they can see their selections and change their available time slots.
4. Timeslots that have already been selected by other users will be shown in a different color to encourage user to choose similar time slots.
5. On the bottom of the page there is the Submit button. By clicking submit, the user submits his/her timeslots to the system. If the user is an admin to the event, there will additionally be a Finalize button. By clicking finalize, the admin finalizes the current event, the system picks the optimal meeting time based on user selection. If the user is a manager, or user, there will be a quit option for the user to remove himself/herself from the current event. All of his previous participation datas (including timeslots they have chosen for themselves will be cleared)

- **Testing**
  - Client/Server
    we plan to test all our pages interactively.
    - Signup Page,
      - we first try to sign up a new account with empty username
      - We try to sign up a new account with empty password
      - We try to sign up a new account with an already-existing username
      - Then we try to input a pair of username and password that do not exit. We are expected to succeed and be directed to the login page.
    - Login Page,
      - we first try to sign up a new account and then log in with correct username and password.
      - Then we try to input a pair of username and password that do not exit. We are expected to see an error message in a pop-up window.
    - Event Page
      - We first try to create an event with already-existing username, an error message is supposed to display
    - Calendar Page
      - Test that admin, managers, and users have different options in their event page
      - Test invite
        - Test that invitation to invalid user triggers error message
        - Test that invitation to users already in the event triggers notifying message
      - Test Set admin
        - Test that setting admin to users not in the event triggers error message
      - We try to select some time slots, refresh the page to test these timeslots are recorded correctly by the user
      - We create an event, have the admin invite a user, and check that when either chooses any timeslots, the other users will be able to see it on their sides.
      - Test that when an event is set to anon, names of users when hovered over a time slot will be displayed as “Anonymous”
      - Test that after admin finalizes an event, the status changes on the page
      - Test that when a user is not logged in, access is denied all the time.

  - **Database**
    - We use OUnit tests to test all the functions requested in the database.mli.
    - Majorly tested the following functions,
      - Test add_new_account

            - Test check_password
            - Test add_new_event
        - Test other functions that are implemented as well with corner cases
        - For every function we tested,
            - we first add  some  accounts and events to the database and then test if our functions can access the database
            - And  then we retrieve the correct information.
            - After testing in OUnit test cases, we use terminal to open the Postgresql database to see if the actual data is consistent with our expectation, i.e, if user id and event id are unique and if the number of accounts, events, and time slots are correct.
- **External Dependency**
    - **PGOcaml**
    - **Eliom**
    - **Unix**
    - **OUnit**
- **Division of Labor**
    - Each of us spent approximately 50-55 hours on this project as Eliom (the library that provided client/server architecture.
    - Division of Labor
        - Wenyuan Ma (wm263): Designed and implemented database based on relational models. Used PGOcaml module to interface with PostgreSQL to create a database called coodle_data, together with related tables with columns of different types. Implemented database functions that can mutate fields in the database (i.e add new account, change admin right) or retrieve information from it (i.e check password, get user's events). Tested the functionality of database module by developing sufficient OUnit test cases which covers all possible situations and potential concurrency issues (i.e if the order of execution would influence the eventual state).
        - Qihang Yan (qy33): Implemented and Designed "Login" page, "Sign up" page that mostly deals with the "account" aspect of the datas stored. Makes sure that all possible error cases potentially caused in this phase, such as wrong password when logging in, signing up with already-existing username. Implemented "My Schedule" page, that attains the list of finalized events for the current user from the database, parses those finalized timeslot results and sorts them in chronological orders. Furthermore, implemented "Create Event Page" by first learning how html input types essentially works, and then understanding how to implement these in Ocaml using the corresponding library Eliom.content (which came with the collective library Eliom)
        - Tianyi Zhang (tz58): Figure out how to compile Eliom (which is non-trivial : ), our Makefile is adapted from several tutorials). Designed and implemented "Event" page, and help other implementation with a

special focus on server-client communication. Experimenting with server functions, javascript (js_of_ocaml), client side cache to offer solution to several pages (Event, signup, My Schedule…) Connecting each part of the project from organizing modules, writing specifications to request database implementation, to helping designing.

- Shiran Chen (sc2289): Designed and implemented the web interface for coodle using Eliom's library and CSS. Register web pages with CSS files using Eliom's module Eliom_tools.F.html, which generates web pages in HTML5. "login.css" corresponds to the interface for login page and event list page. "coodle.css" corresponds to the interface for signup page, calendar page and invite/add manager pages. To create a menu bar at the top of the web page, used an outer div registered with a class name and contents of the menu as inner divs registered with the same class name in the module Eliom_registration.App. Implemented the invite service that adds a valid user to the event.

- **Demos**

Below we have included a few screenshots of some of the key features included in our project as demos for their prospective usages.

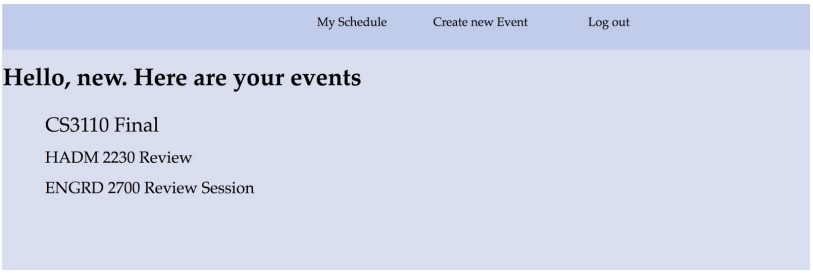
- Log in

  The user can either log in to their existing account or create a new one by clicking on the "Create a new account" link at the bottom


  All of the account informations are safely stored on our database. And when a user enters an invalid username or password, an error message will display to warn the user.

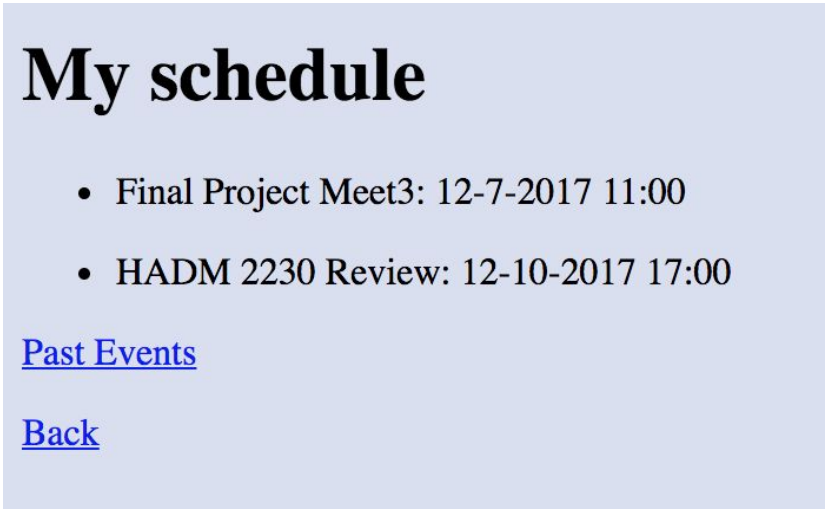
- My Events Page

  After the user successfully logs into his/her account, there will be the list of events the user is currently active in,




Please notice at the top of this page, there are two other options, one is to create a new event, and the other is look at his schedule.

- My Schedule Page



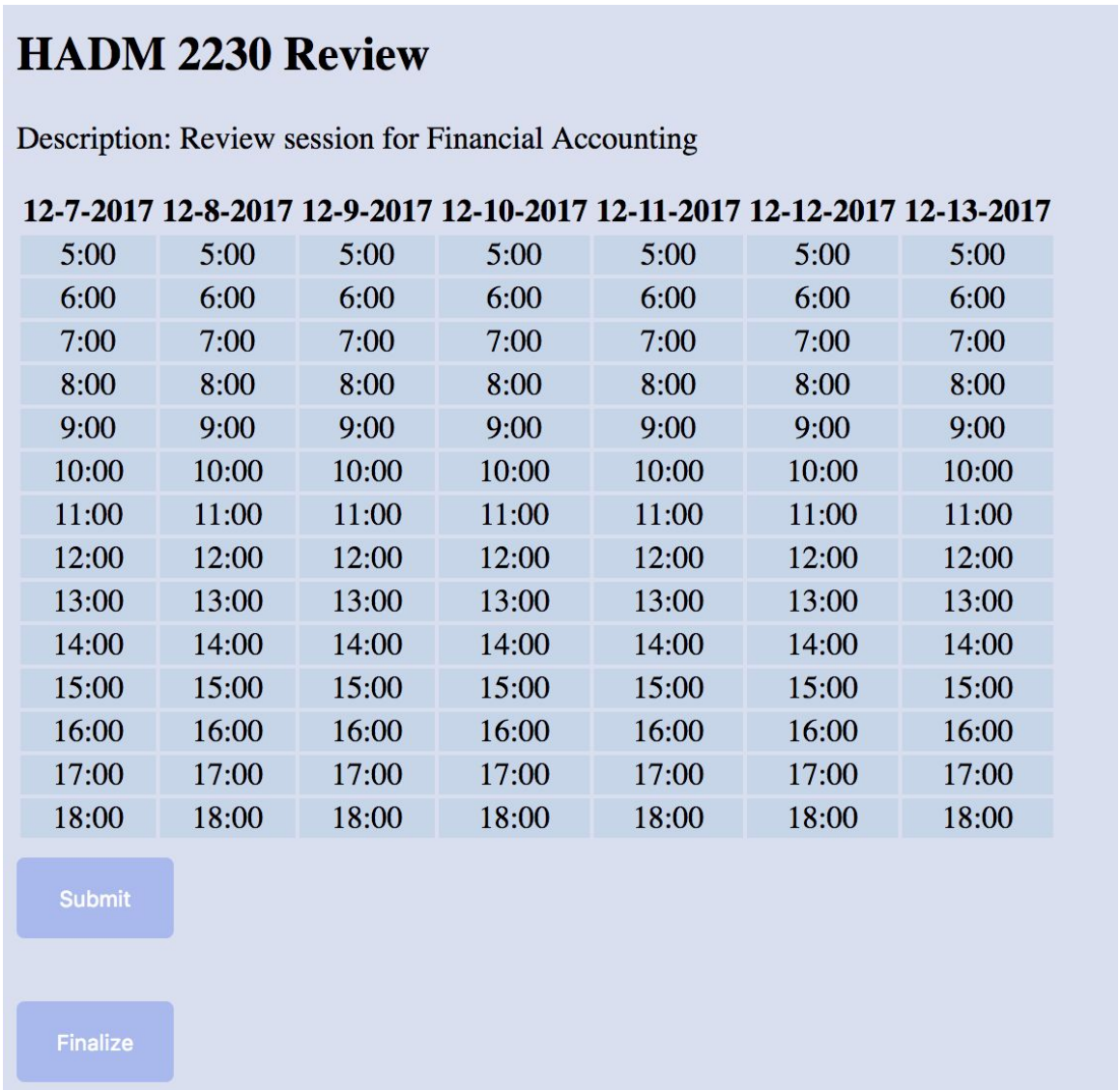

The “My Schedule” feature allows the user to see the list of all finalized event he/she is participating in, in a chronological order. Furthermore, only upcoming events will be shown in this page, but if the user would like to check back at the log of any previous events she/he has participated in, the user is welcomed to do so by clicking “Past Events.” For Example,


- Open Event Page

At the top of the page, the user will be able to see the following menu bar.

This is the menu bar with options strictly limited to admins, and will be different for users of other rights.


Below the menu bar, is the timeslots table. We used Hover Design as we tried to make the page look more “Modern” and appeal to college students who are our major targets.




Once other users submitted their timeslot, those slots will be displayed in different colors. And next to the table, there will be a list view of the participants in the event (or “Anonymous” by setting) and the number of participants who have chosen this slot to encourage users to select popular slots as best as they can.

## *Implementation Progress Record:*

**Week 11/25**

1. **Finish implementing Login page**
    a. **Display Error message if Password is wrong/User DNE (using eref)**
    b. **Redirect user to Event List is Login is determined to be successful**
2. **Finish implementing Signup page**
    a. **Display Error message if Username has already been taken (stay at same page)**
    b. **Redirect User to Login page is Sign up is determined to be successful and display a success message to the user at Login Page**
3. **Finish implementing Event List page**
    a. **Displays the list of events using hyperlinks**
    b. **Each event has its own page with event/eventname as the suffix**
    c. **Display "Access Denied" if current user does not have access to event/eventname (*Need to implement check user right feature*)**
    d. **Display "Event Does not exist" if event with eventname does not exist in database.**