★ User-Content Management

★ Our application focuses on scheduling meetings between friends, basic functionalities like user registration and login are provided.

★ After the user logs into their account, they can create a schedule for a meeting by specifying the details about the meeting.

★ The schedule option will have different fields that are to be updated by the user to give information to other users about the meeting.

★ A mapping is done between the Users and Schedules, where a User can have multiple Schedules and a Schedule can have multiple Users.

★ We can see a many-to-many relationship between the Users and Schedule.

★ Specify what kind of information each of these areas will have in your application.

★ User model will have the basic sign-in, sign-up pages to login and register.

★ After logging in the user can see any previously scheduled meetings or create a new one.

★ For creating a new schedule the user should specify the information about the meeting like what is it about, where etc.,

★ Schedule model will have the fields of title, place, note, date, time, comments section and options to choose from.

★ Title will specify what is the meeting about, Place will give us the location of the meet, Note will have any messages of greetings to the people the user wants to send.

★ Date and Time can be mentioned in the Calendar which give information about the day and what time the meeting will be held. A comments section is also provided where the users can give in comments or replies to other users posts.

★ We also provide options for the users to choose about the meeting like yes meaning will attend the meeting, no and maybe.

★ The user who creates can the schedule can invite different users using their email addresses and also the user gets notified about any updated information about the meeting such as a user has accepted for the meeting or declined for the meeting etc.,

★ Avoiding SQL injection and XSS attacks

Injection is a class of attacks that introduce malicious code or parameters into a web application in order to run it within its security context. Prominent examples of injection are cross-site scripting (XSS) and SQL injection.

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

To mount SQL injection attacks

- The attacker needs to have information about the data schema
- The application needs to not validate input data

So to prevent SQL injection attacks, we can

- Filter all input data for special characters and out-of-range values;
- Use different database credentials for different uses (e.g. don't allow the application server user to execute the DROP command)
- Always use parameterized prepared statements.

Cross-site Scripting (XSS)

In cross-site scripting (XSS) attacks, the attacker places fraudulent code on your site through user-entered data.

Possible reasons for XSS attacks:

- Stealing cookies
- Redirecting users to a malicious site
- Presenting fake content convincing the user to enter private data

So to prevent XSS attacks the countermeasures are

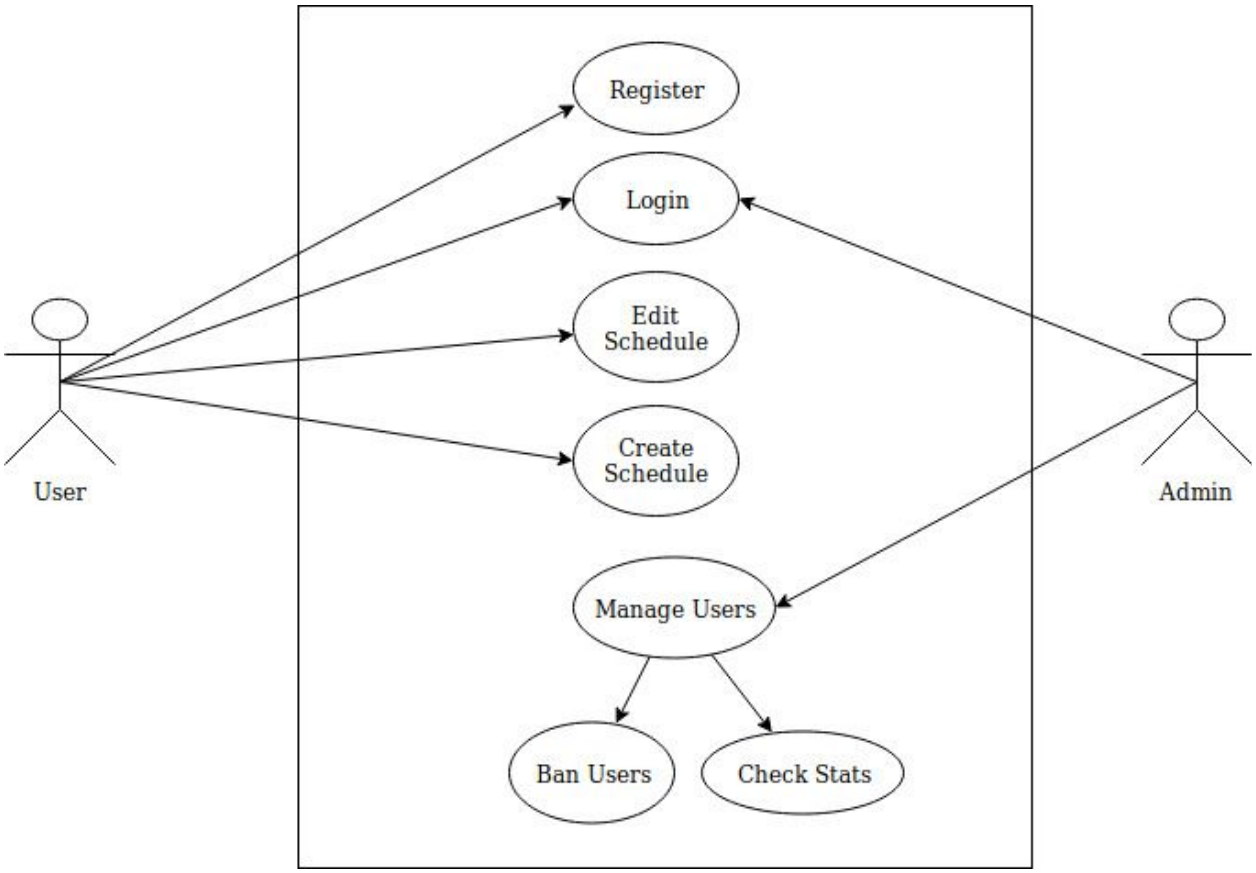
- Filter malicious input : whitelist input filtering instead of blacklist.
- Escape the output of the web application: Escape all output of the application, especially when re-displaying user input, which hasn't been input-filtered (as in the search form example earlier on).

★ Audit your software for any sensitive data like database passwords make sure they are secure. Document the result of your audit and plan

- Keep the database password out of version control
  - ➢ create a user named *scheduler*(our project name) and a file to hold the secrets
  - ➢ Inside that file, we provide the database passwords
  - ➢ Then we add it to our *application.rb*
  - ➢ Run Passenger as our application's user(*scheduler*) .
  - ➢ Thereafter we can access the database passwords from anywhere in our application, i.e. in the *database.yml*

- By default, Rails logs all requests being made to the web application. But log files can be a huge security issue, as they may contain login credentials, credit card numbers etc.So in the event an attacker got (full) access to the web server, encrypting secrets and passwords in the database will be quite useless, if the log files list them in clear text. We can filter certain request parameters from log files by appending them to config.filter_parameters in the application configuration. These parameters will be marked [FILTERED] in the log.
  config.filter_parameters << :password

★ User Data Model



User
+ email: string
+ password: string
+ is_admin? : bool = false

★ **Sketch user registration and management page flow.**

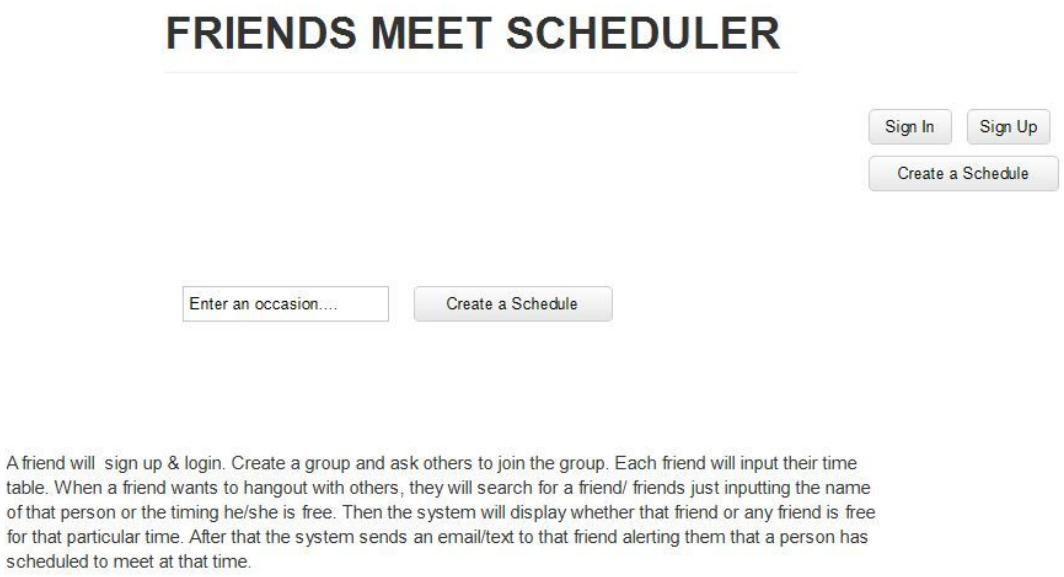


Figure 1 : Homepage

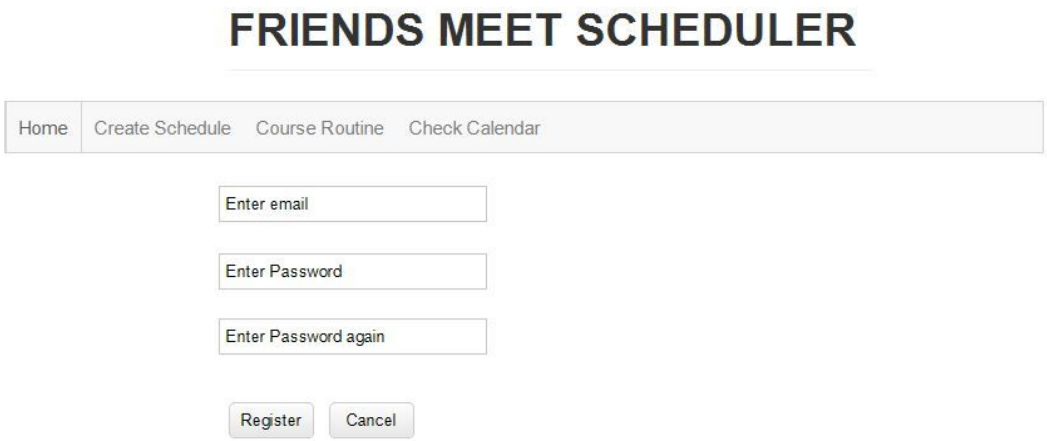


Figure 2 : Registration Page

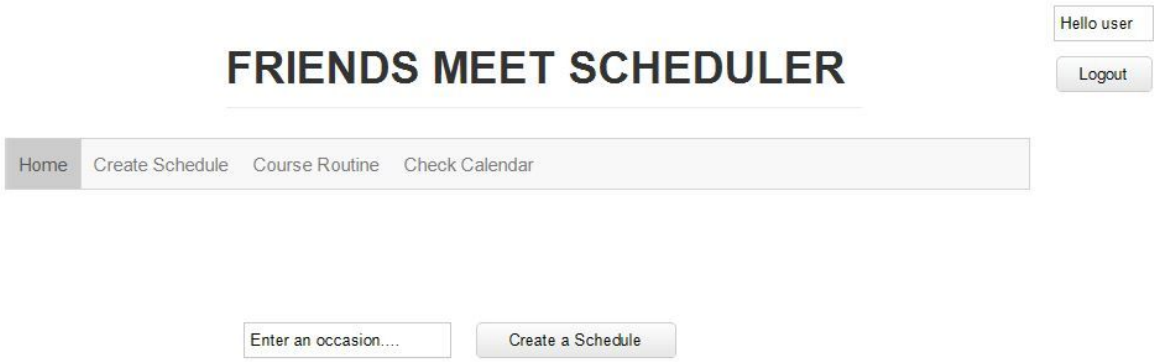


Figure 3 : Successful Authentication

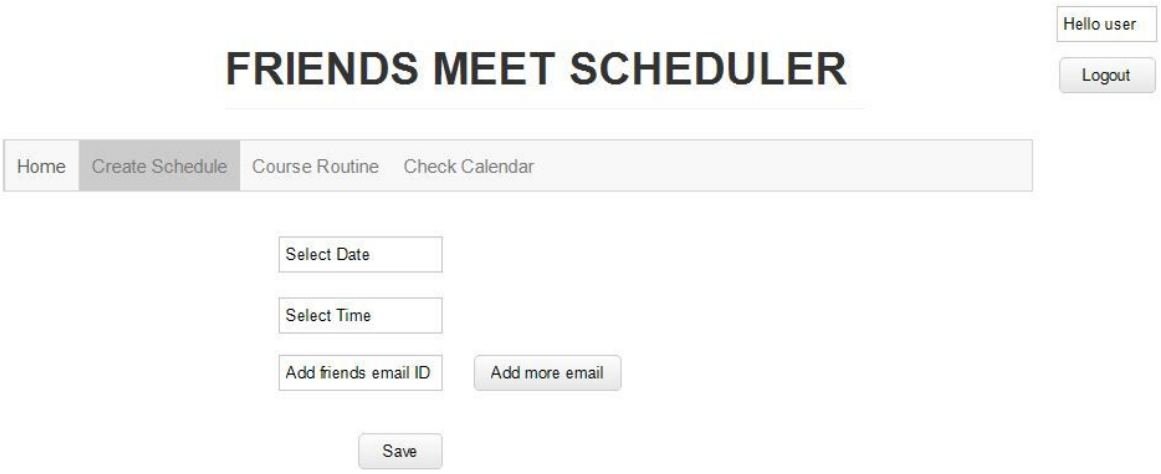


Figure 4 : Create a schedule

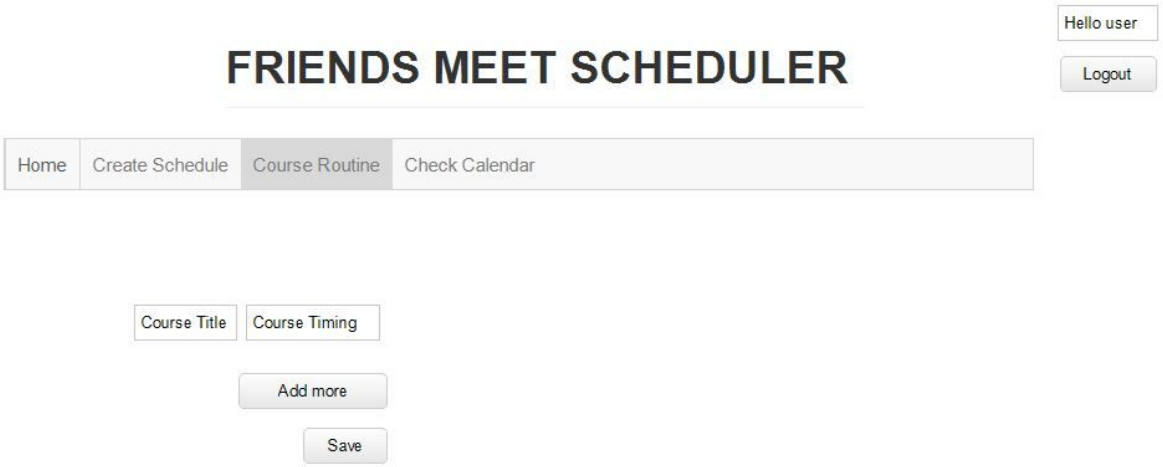


Figure 5 : Adding user's course routine

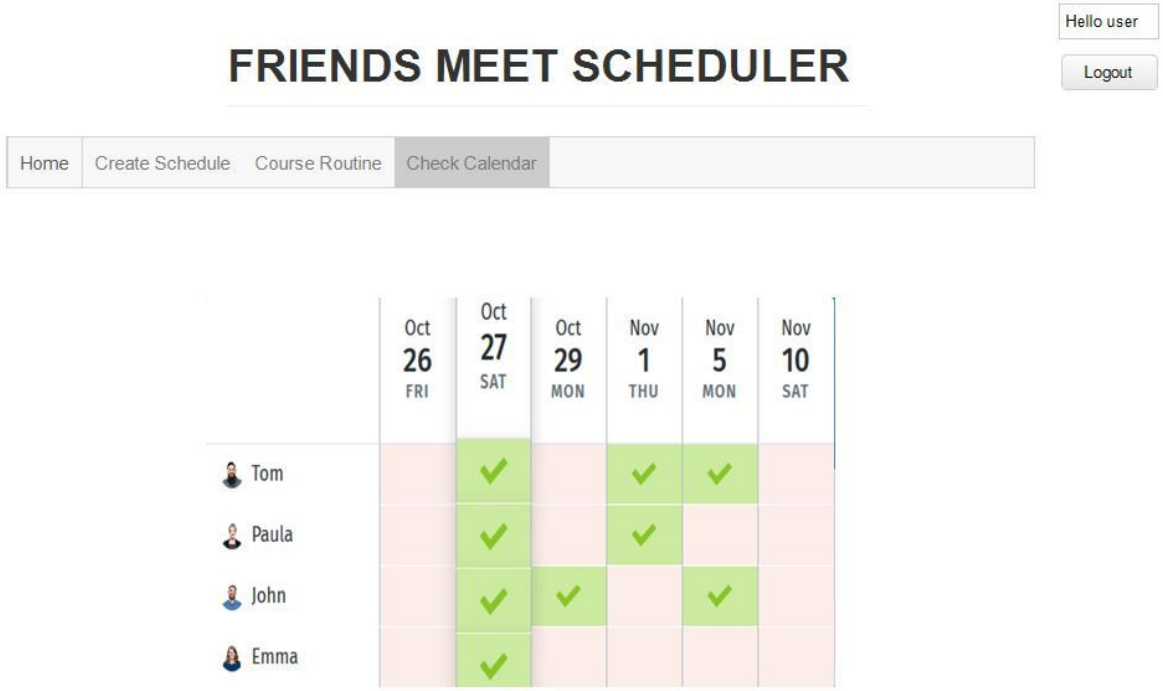


Figure 6 : Checking calendar

★ UAT Test

Feature: Registration
Scenario: User should be able to register

```
        Given I am on the homepage
        And I am not signed up
        Then I should see a link to sign up
        When I click on the sign up link
        Then I should see a sign up form to fill
        When I submit the sign up form
        Then I should see the homepage

Given(/^I am on the homepage$/) do
        visit '/'
end

And (/^I am not signed up$/) do
   @user = FactoryBot.create :user

end

Then(/^I should see a link to sign up$/) do
        find_link('Sign up', href:"/users/sign_up")
end

When (/^I click on the sign up link$/) do
        find_link('Sign up', href:"/users/sign_up").click
end

Then (/^I should see a sign up form to fill$/) do
   expect(page).to have_selector('form#new_user')
end

When (/^I submit the sign up form$/) do
        fill_in 'Email', with: @user.email
        fill_in 'Password', with: @user.password
   fill_in 'Password confirmation', with: @user.password_confirmation
end
```

```
Then (/^I should see the homepage$/) do
        visit root_path
end
```

Feature: Login
Scenario: User should be able to login

```
        Given I am on the home page
        And I am not logged in
        Then I should see a link to sign in
        When I click on the sign in link
        Then I should see a form to fill
        When I submit the sign in form
        Then I should be on the users home page

Given(/^I am on the home page$/) do
  visit '/'
end

And (/^I am not logged in$/) do
  @user = FactoryBot.create :user
end

Then(/^I should see a link to sign in$/) do
  find_link('Sign in', href:"/users/sign_in")
end

When (/^I click on the sign in link$/) do
  find_link('Sign in', href:"/users/sign_in").click
end

Then (/^I should see a form to fill$/) do
  expect(page).to have_selector('form#new_user')
end

When (/^I submit the sign in form$/) do
  fill_in 'Email', with: @user.email
  fill_in 'Password', with: @user.password
end
```

```
Then (/^I should be on the users home page$/) do
  visit root_path
End
```

Feature: Admin Page
Scenario: See Registered Users

```
    Given I am an Admin
    And I am in the User Admin Page
    Then I should see user email and option to ban or unban the user
    When I click the ban user button
    Then I should see an option to unban the user
```

```
Given (/^I am an Admin$/) do
  @user = FactoryBot.create :user
end

And (/^I am in the User Admin Page$/) do
  visit '/useradminister/index'
end

Then (/^I should see user email and option to ban or unban the user$/) do
  expect(page).to have_content "#{@user.email}"
end

When (/^I click the ban user button$/) do
  find_link('Ban The User', href: "/useradminister/ban?email").click
end

Then (/^I should see an option to unban the user$/) do
  find_link('Unban The User', href: "/useradminister/unban?email")
end
```

★ References

1. https://cis.ait.asia/course_offerings/1/lecture_notes/6
2. https://guides.rubyonrails.org/security.html
3. https://medium.com/craft-academy/encrypted-credentials-in-ruby-on-rails-9db1f36d8570
4. https://www.w3schools.com/sql/sql_injection.asp
5. http://rubyrobot.github.io/blog/2014/04/08/secure-rails-deployment-and-passwords-best-practices/