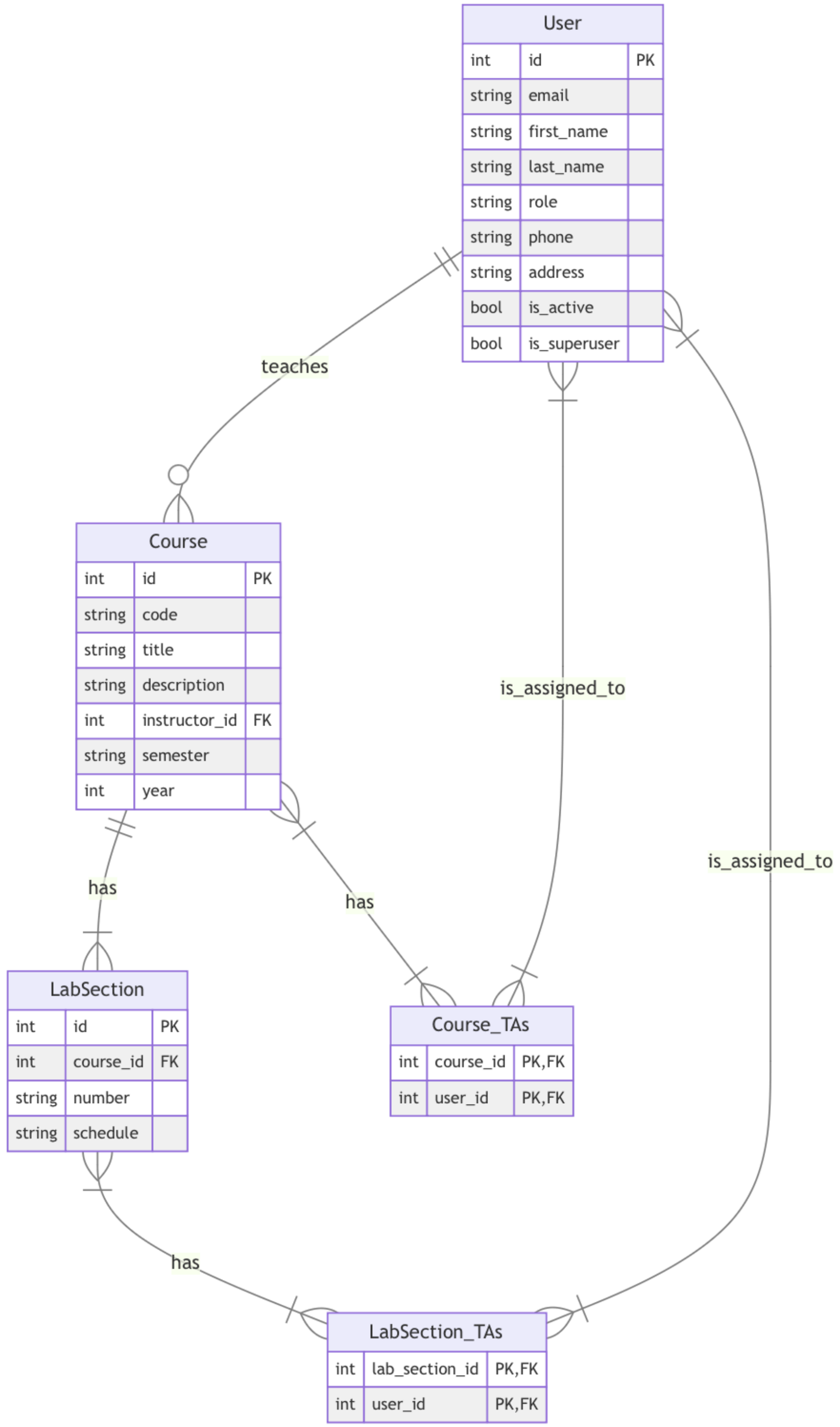
Database Design



User

int	id	PK
string	email	
string	first_name	
string	last_name	
string	role	
string	phone	
string	address	
bool	is_active	
bool	is_superuser	

Course

int	id	PK
string	code	
string	title	
string	description	
int	instructor_id	FK
string	semester	
int	year	

LabSection

int	id	PK
int	course_id	FK
string	number	
string	schedule	

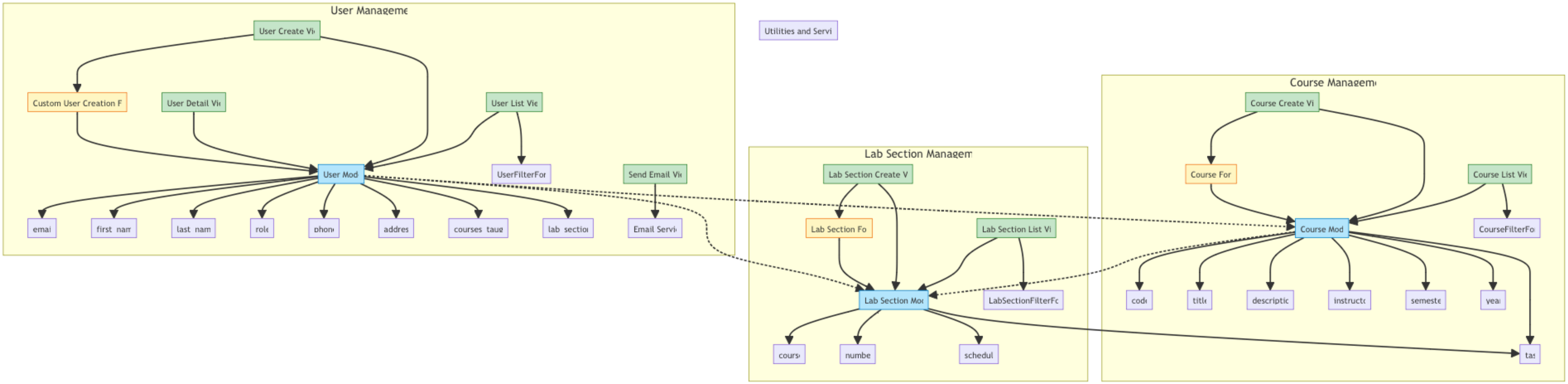
Course_TAs

int	course_id	PK,FK
int	user_id	PK,FK

LabSection_TAs

int	lab_section_id	PK,FK
int	user_id	PK,FK

Software Design



Method Descriptions:

course_management/views.py

Method: CourseDetailView.get_context_data(**kwargs)

Preconditions: The view must be accessed by a user who is logged in due to LoginRequiredMixin. self.object must be a valid Course instance that exists in the database.

Postconditions: Returns the context data for the view, which includes the course object itself and a queryset of LabSection objects related to the course.

Side-effects: None; the method queries the database but does not alter any data.

- kwargs (in-out): Keyword arguments that might be used to modify the context or pass additional data to the template.
- self (in): Used to access instance variables including the object which is the Course instance being viewed.

Method: CourseListView.get_queryset()
Preconditions: The view must be accessed by a user who is logged in due to LoginRequiredMixin.
Postconditions: Returns a queryset of Course objects potentially filtered based on the search query parameters included in the request (title, code, description).
Side-effects: None; it performs a read-only database query.
- self (in): Utilized to access class-level attributes and methods, such as retrieving the request object to check for GET parameters.

Method: CourseCreateView.test_func()
Preconditions: User must be logged in, inheriting from LoginRequiredMixin and UserPassesTestMixin.
Postconditions: Determines if the user can access the course creation form based on their role or superuser status.
Side-effects: Controls access to the course creation functionality; does not modify any data directly.
- self (in): Used to check the current user's role and superuser status.

Method: CourseUpdateView.get_success_url()
Preconditions: The course update process must have been completed successfully.
Postconditions: Returns the URL to which the user will be redirected after successfully updating a course.
Side-effects: None; purely provides URL for redirection.
- self (in): Utilized to access the pk of the updated Course object for generating the redirect URL.

Method: CourseUpdateView.test_func()
Preconditions: User must be logged in. Only users with a 'Supervisor' role or superusers are allowed to proceed.
Postconditions: Returns True if the user is allowed to update the course, False otherwise.
Side-effects: Controls access to the course update functionality.
- self (in): Used to verify the user's role and superuser status.

Method: CourseDeleteView.test_func()

Preconditions: User must be logged in. Only users with a 'Supervisor' role or superusers can initiate a delete operation.

Postconditions: Determines whether the user is permitted to delete a course.

Side-effects: Controls who can delete courses, affecting data integrity and access controls.

- self (in): Utilized to check the user's role and superuser status for authorization.

course_management/forms.py

Method: CourseForm.__init__(self, *args, **kwargs)
Preconditions: This method is called when an instance of CourseForm is created.
Postconditions: Initializes the form with specified fields and widgets, and modifies the queryset for the instructor and tas fields to filter based on user roles.
Side-effects: Modifies the form instance by setting the queryset for instructor and tas fields.
- args (in): Positional arguments passed to the form instance. - kwargs (in-out): Keyword arguments that might include an instance for updating or initial data for the form fields.

Method: CourseForm.clean_code(self)
Preconditions: clean_code is called as part of the form validation process, assuming that code field data has been input.
Postconditions: Returns the cleaned code data which is uppercased and validated to be alphanumeric.
Side-effects: Raises a ValidationError if the code is not alphanumeric, affecting the form's validation process.
- self (in): Used to access the form field code.

Method: CourseForm.clean_year(self)
Preconditions: clean_year is called during form validation, assuming that the year field has been populated.
Postconditions: Returns the cleaned year data after ensuring it is greater than or equal to 1900.
Side-effects: Raises a ValidationError if the year is less than 1900, which directly impacts the form's overall validity.
- self (in): Utilized to access the year field data from the form.

course_management/models.py

Method: Course.__str__(self)
Preconditions: Assumes that a Course instance has been properly initialized and populated with the necessary fields (code and title).
Postconditions: Returns a string representation of the Course instance, formatted as "code - title”.
Side-effects: None; this method only formats and returns a string representation of the object and does not alter any object state.
- self (in): Used to access the code and title attributes of the Course instance.

course_management/amin.py

Method: CourseAdmin.instructor_name(self, obj)
Preconditions: Assumes that obj is an instance of the Course model.
Postconditions: Returns the email of the instructor associated with the course if an instructor is assigned, otherwise returns 'No instructor assigned’.
Side-effects: None; this method only retrieves data and does not modify the state of the Course instance or any other part of the application.
- self (in): The instance of CourseAdmin using which this method is called. - obj (in): The Course instance for which the admin interface needs the instructor's email. -

user_management/views.py

Method: UserCreateView.form_valid(form)
Preconditions: The form passed must be an instance of CustomUserCreationForm and must be valid.
Postconditions: A new User instance is created and saved to the database. The method then redirects to the URL defined by self.success_url.
Side-effects: A new user record is added to the database.
- form (in): The form instance containing the validated data for the new user.

Method: UserUpdateView.test_func()
Preconditions: User must be logged in due to LoginRequiredMixin.
Postconditions: Returns True if the current user is a superuser or the user id matches the id in the URL (self.kwargs['pk']); otherwise, returns False.
Side-effects: Determines whether the user has permission to proceed with updating a user record.
- self (in): Used to access the current user and URL parameters.

Method: UserListView.get_queryset()
Preconditions: User must be logged in due to LoginRequiredMixin.
Postconditions: Returns a queryset of User objects that may be filtered based on a search query present in the request's GET parameters.
Side-effects: None; performs a read-only operation on the database.
- self (in): Used to access the request object to retrieve GET parameters.

Method: UserDetailView.test_func()
Preconditions: User must be logged in.
Postconditions: Returns True if the current user is a superuser, the role includes 'TA', 'Instructor', or 'Supervisor', or the user id matches the id in the URL; otherwise, returns False.
Side-effects: Controls access to viewing detailed user information.
- self (in): Used to access the current user and URL parameters.

user_management/forms.py

Method: CustomUserCreationForm.__init__(self, *args, **kwargs)
Preconditions: The form is instantiated typically during a user creation process.
Postconditions: Initializes a user creation form with custom settings for the email field and role choices.
Side-effects: Sets the email field to required and specifies role choices.
- args (in): Positional arguments for form initialization. - kwargs (in-out): Keyword arguments that might include initial data or instance linking.

Method: CustomUserUpdateForm.__init__(self, *args, **kwargs)
Preconditions: The form is instantiated typically during a user update process.
Postconditions: Initializes a user update form where the email field is disabled.
Side-effects: Disables the email field to prevent it from being edited.
- args (in): Positional arguments for form initialization. - kwargs (in-out): Keyword arguments that might include initial data or instance linking.

Method: CustomPasswordChangeForm.clean(self)
Preconditions: All fields (old_password, new_password, confirm_password) are populated.
Postconditions: Validates that the new password and confirm password fields match. If they do not, a validation error is added to the form.
Side-effects: Can raise a validation error which may affect the form's validity in the context of the view processing the form submission.
- self (in): Used to access and manipulate form data.

user_management/models.py

Method: UserManager.create_user(self, email, password=None, **extra_fields)

Preconditions: An email must be provided. If not, a ValueError is raised.

Postconditions: A new User instance is created with the provided email, password, and additional fields. The password is hashed before saving.

Side-effects: A new user record is added to the database.

- email (in): The email address for the new user, required for creating the account.
- password (in, optional): The raw password, which will be hashed upon saving.
- extra_fields (in-out, optional): A dictionary of additional fields to be set on the user object.

Method: UserManager.create_superuser(self, email, password, **extra_fields)

Preconditions: is_superuser must be set to True in extra_fields; otherwise, a ValueError is raised.

Postconditions: A new superuser instance is created with the provided credentials and fields.

Side-effects: A new superuser record is added to the database.

- email (in): The email address for the new superuser.
- password (in): The raw password for the superuser, which will be hashed upon saving.
- extra_fields (in-out): Additional fields to be set on the superuser object, with is_superuser ensured to be True.

Method: User.get_full_name(self)

Preconditions: None.

Postconditions: Returns the full name of the user, combining first_name and last_name. If both are empty, the email is returned.

Side-effects: None.

- self (in): The user instance from which to retrieve the name.

Method: User.get_short_name(self)
Preconditions: None.
Postconditions: Returns the first_name of the user.
Side-effects: None.
- self (in): The user instance from which to retrieve the short name.

Method: User.is_staff
Preconditions: None.
Postconditions: Returns True if the user is a superuser, indicating they have staff status.
Side-effects: None.
- self (in): The user instance being queried for staff status.

user_management/signals.py

Method: user_created_or_updated(sender, instance, created, **kwargs)

Preconditions: This function is designed to be connected to the post_save signal of the User model.

Postconditions: Sends an email to the user when their account is created or updated. The content of the email differs based on whether the account was just created or updated.

Side-effects: Sends an email to the user's email address, which could fail silently if configured to do so.

- sender (in): The model class that sent the signal; expected to be User.
- instance (in): The instance of User that was saved.
- created (in): A boolean indicating whether the User instance was created (True) or updated (False).
- kwargs (in-out): Additional keyword arguments passed along with the signal.

user_management/admin.py

Method: UserCreationForm.clean_password2(self)
Preconditions: password1 and password2 fields must be filled in the form.
Postconditions: Validates that the two password fields match. If they don't, it raises a validation error.
Side-effects: Raises a ValidationError if the passwords do not match, which prevents the form from being valid.
- self (in): The form instance containing the password fields.

Method: UserCreationForm.save(self, commit=True)
Preconditions: Assumes that the form is valid.
Postconditions: Saves the new User with a hashed password if commit is True. Returns the User instance.
Side-effects: Modifies the database by adding a new User record if commit is True.
- commit (in, optional): A boolean that determines whether to save the user immediately or not.

Method: UserAdmin.get_form(self, request, obj=None, **kwargs)
Preconditions: Called when rendering the form in the admin interface.
Postconditions: Returns UserCreationForm if creating a new user (obj is None) or UserChangeForm for editing an existing user.
Side-effects: None; selects the appropriate form based on the user object being None or not.
- request (in): The HTTP request object. - obj (in, optional): The User instance being edited, if any.

core/views.py

Method: home(request)
Preconditions: The function expects an HTTP request object.
Postconditions: Renders and returns the home.html template as an HTTP response.
Side-effects: None; this function only generates an HTTP response with the specified template.
- request (in): The HTTP request object containing metadata and data about the request.

Method: search_select(request)
Preconditions: The function is called with an HTTP request object.
Postconditions: Renders and returns the search_select.html template as an HTTP response.
Side-effects: None; like home, it only returns a rendered response.
- request (in): The HTTP request object, as it comes into the view function.

core/forms.py

Method: CustomAuthenticationForm.__init__(self, *args, **kwargs)

Preconditions: The form is instantiated typically during the user authentication process.

Postconditions: Initializes the authentication form and modifies the username field to use an email field with autofocus enabled, and labels it as "Email".

Side-effects: Adjusts the default behavior and presentation of the username field in the authentication form.

- args (in): Positional arguments passed to the form, which are forwarded to the superclass initializer.
- kwargs (in-out): Keyword arguments that might include initial data or configurations for the form fields.

lab_section_management/views.py

Method: LabSectionListView.get_queryset()
Preconditions: User must be logged in due to LoginRequiredMixin.
Postconditions: Returns a queryset of LabSection objects potentially filtered based on the search query parameters included in the request (course__title or number).
Side-effects: None; performs a read-only operation on the database.
- self (in): Used to access the request object to retrieve GET parameters.

Method: LabSectionCreateView.test_func()
Preconditions: User must be logged in, inheriting from LoginRequiredMixin and UserPassesTestMixin.
Postconditions: Returns True if the user has the role of 'Supervisor' or 'Instructor', allowing the creation process to proceed.
Side-effects: Determines access control for creating a lab section; does not modify any data.
- self (in): Used to check the current user's role.

Method: LabSectionDetailView.test_func()
Preconditions: User must be logged in.
Postconditions: Always returns True, allowing all authenticated users to view lab section details.
Side-effects: None; merely checks for permission without changing any data.
- self (in): Used to determine access rights based on user authentication.

Method: LabSectionUpdateView.test_func()
Preconditions: User must be logged in and must have either the 'Supervisor' role or be the instructor associated with the lab section.
Postconditions: Returns True if the user meets the role requirements, otherwise returns False.
Side-effects: Determines access control for updating a lab section.
- self (in): Used to access the current user's role and compare it with the lab section's instructor.

Method: LabSectionDeleteView.test_func()

Preconditions: User must be logged in and have authorization to delete the lab section.

Postconditions: Returns True if the user is a 'Supervisor' or the course instructor, allowing deletion.

Side-effects: Controls who can delete a lab section.

- self (in): Used to access the current user's role and the associated course instructor's details.

lab_section_management/forms.py

Method: LabSectionForm.__init__(self, *args, **kwargs)

Preconditions: The form is instantiated typically during the lab section creation or update process.

Postconditions: Initializes the lab section form with specific widgets and help texts, and modifies the queryset for the tas field to filter only users with the 'TA' role.

Side-effects: Adjusts the form fields, particularly the tas field to restrict choices to users with a 'TA' role.

- args (in): Positional arguments for form initialization.
- kwargs (in-out): Keyword arguments that might include initial data or instance linking for form setup.

Method: LabSectionForm.clean_number(self)

Preconditions: number field must be filled in the form.

Postconditions: Returns the cleaned number data which is validated to be alphanumeric and converted to uppercase.

Side-effects: Raises a ValidationError if the number is not alphanumeric, affecting the form's overall validity.

- self (in): The form instance containing the number field.

lab_section_management/models.py

Method: LabSection.__str__(self)

Preconditions: Assumes that a LabSection instance has been properly initialized and populated with the necessary fields (course, number, schedule).

Postconditions: Returns a string representation of the LabSection instance, formatted as "<course code> - Lab <number> (<schedule>)".

Side-effects: None; this method only formats and returns a string representation of the object and does not alter any object state.

- self (in): Used to access the course, number, and schedule attributes of the LabSection instance.

lab_section_management/admin.py

Method: LabSectionAdmin.display_tas(self, obj)

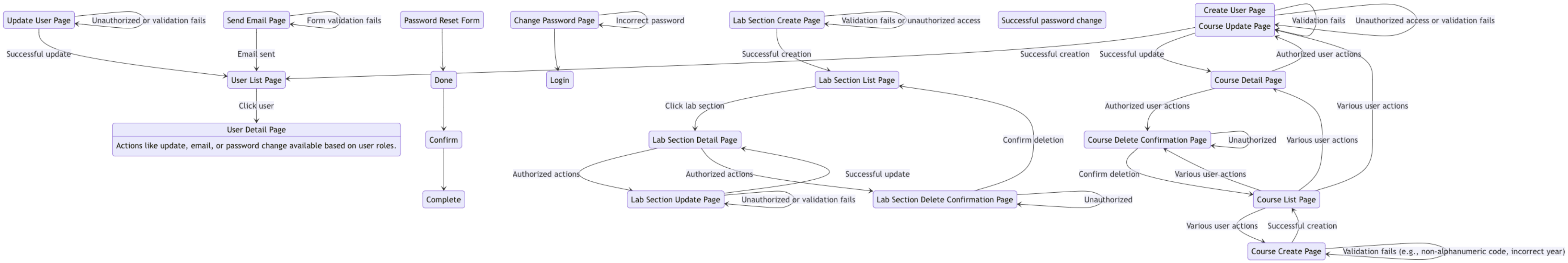
Preconditions: Assumes obj is an instance of the LabSection model.

Postconditions: Returns a string representation of all teaching assistants (TAs) assigned to a lab section, formatted as a comma-separated list of their email addresses.

Side-effects: None; this method performs a read operation on the tas related objects and does not alter any data.

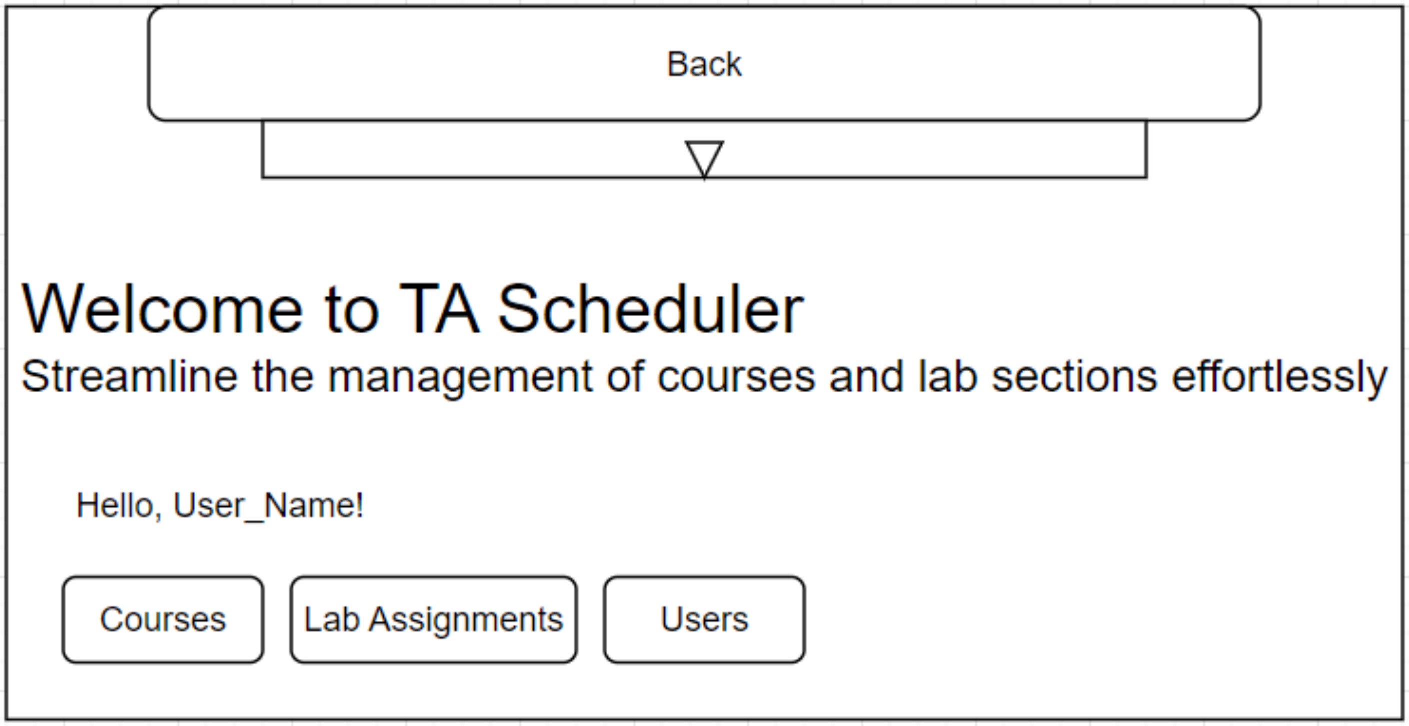
- obj (in): The LabSection instance for which the list of TAs is being formatted.
- self (in): The instance of LabSectionAdmin using which this method is called.

Machine State Diagram:



Prototypes:

Home Page

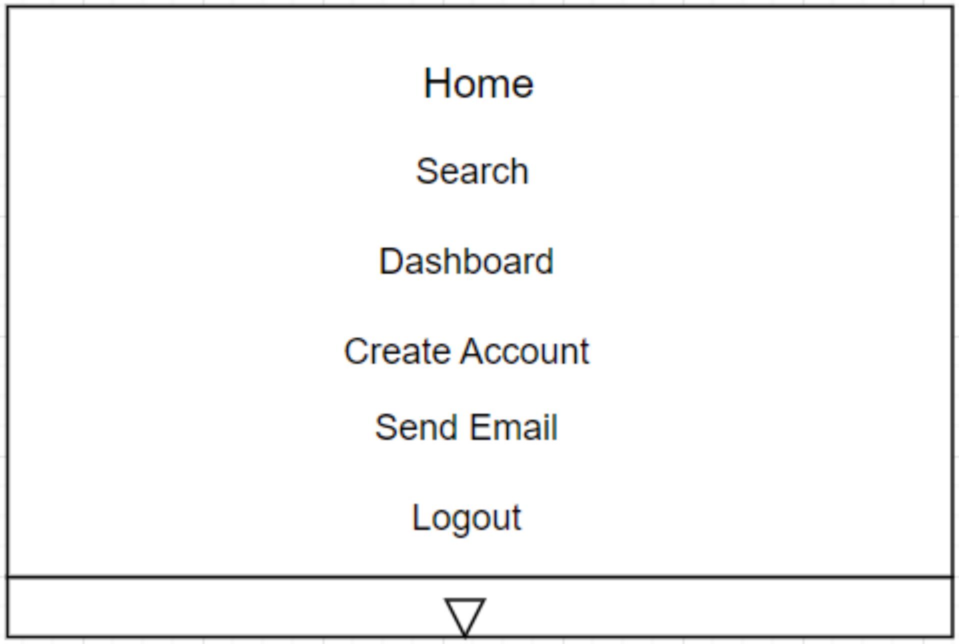


Welcome to TA Scheduler

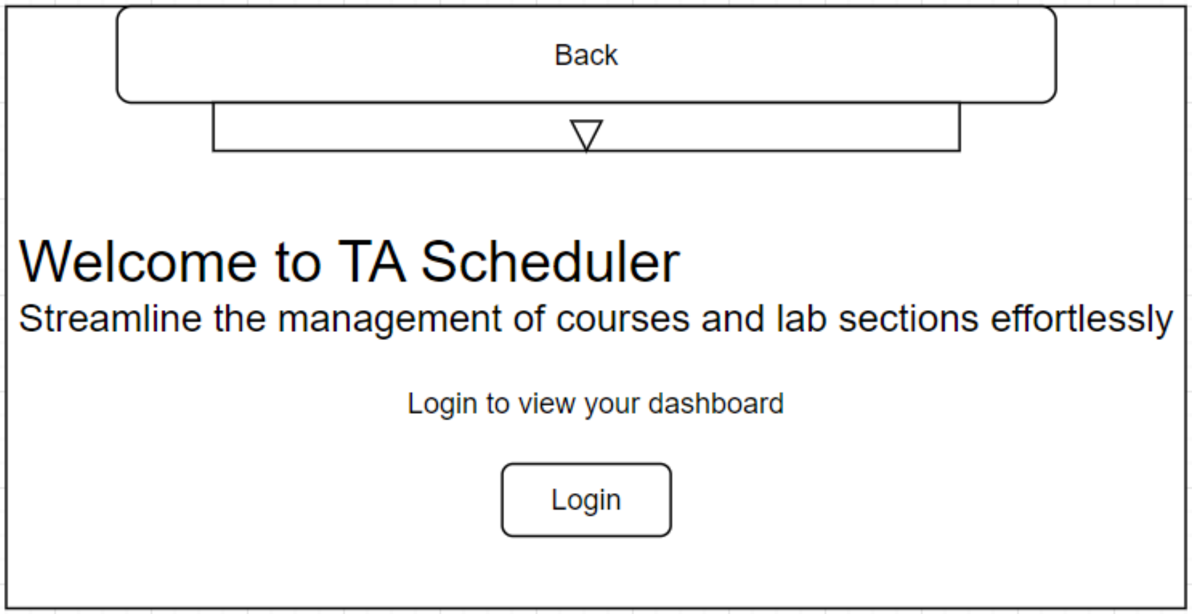
Streamline the management of courses and lab sections effortlessly

Hello, User_Name!

Navigation Bar



Start-Up Page

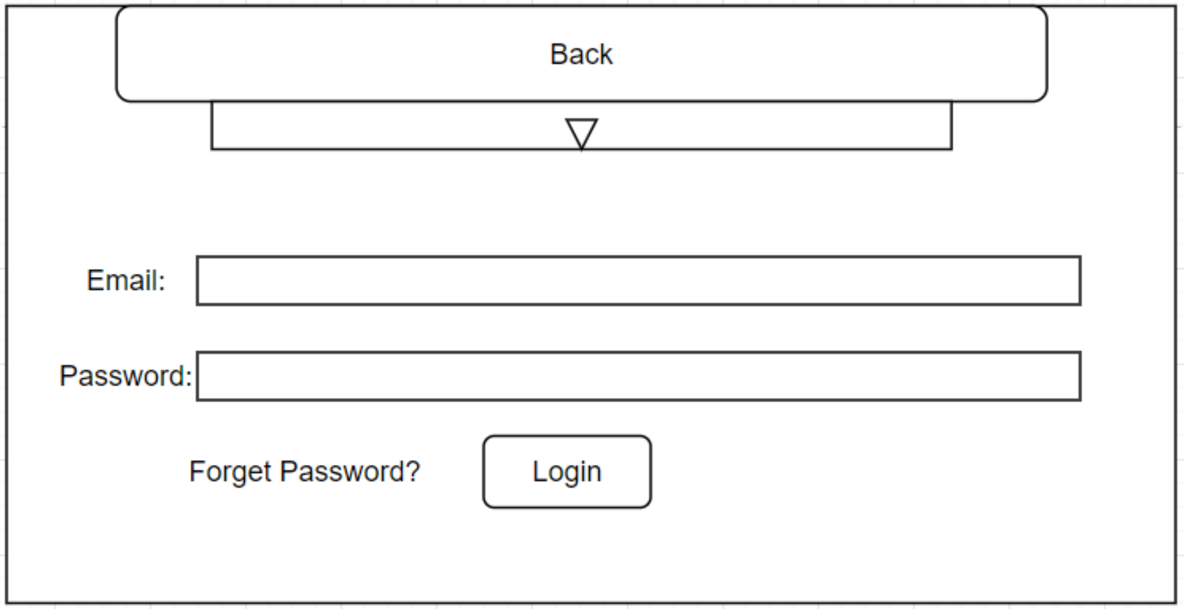


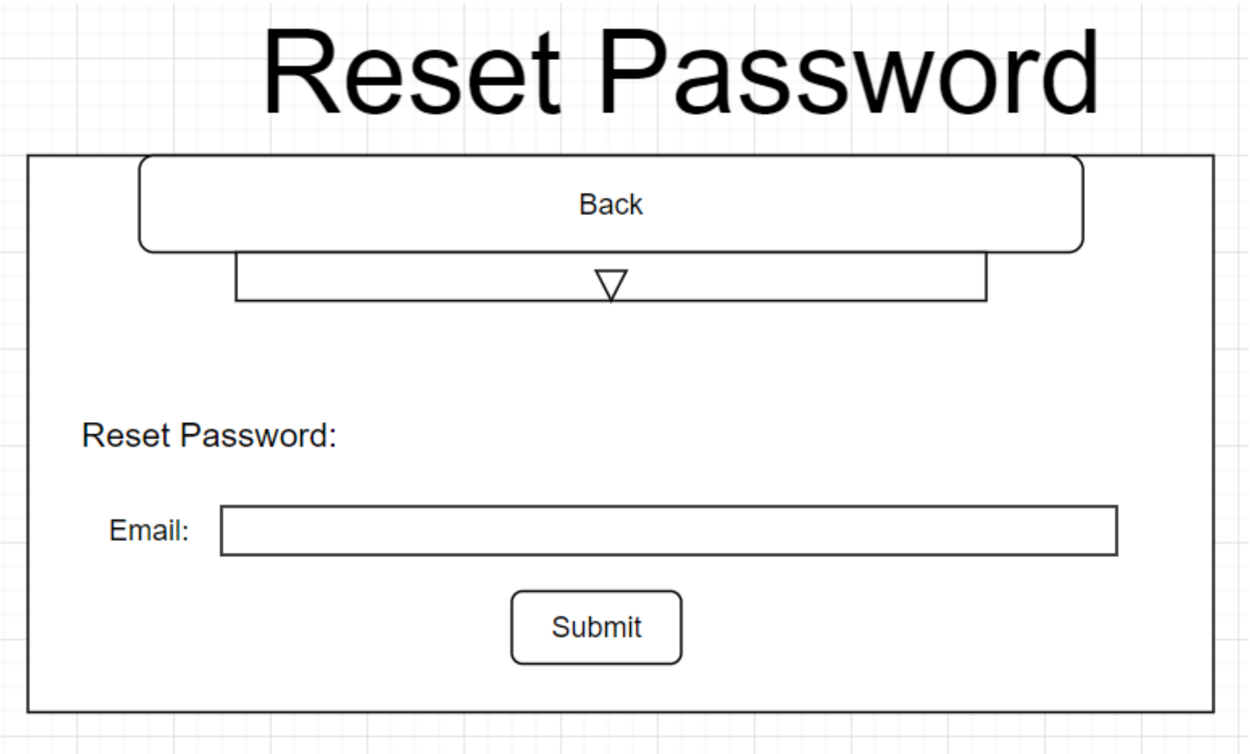
Welcome to TA Scheduler

Streamline the management of courses and lab sections effortlessly

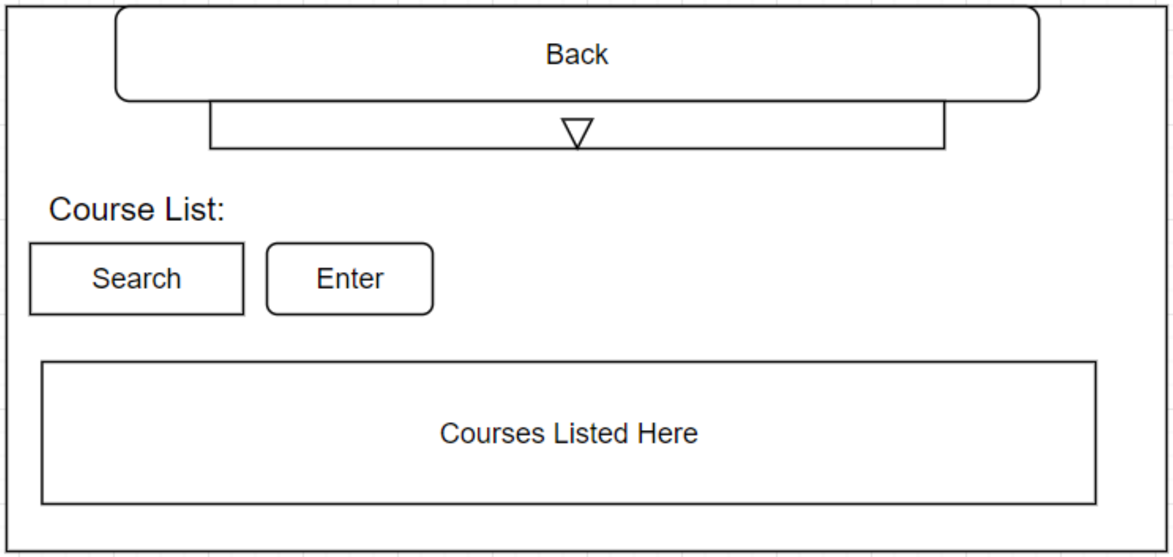
Login to view your dashboard

Login Page

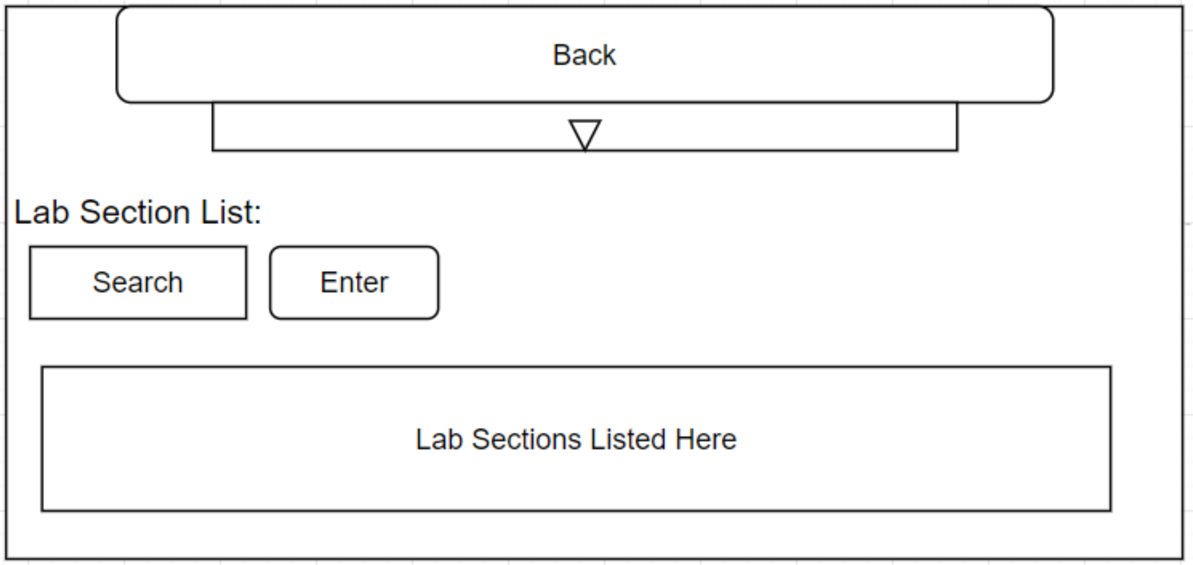


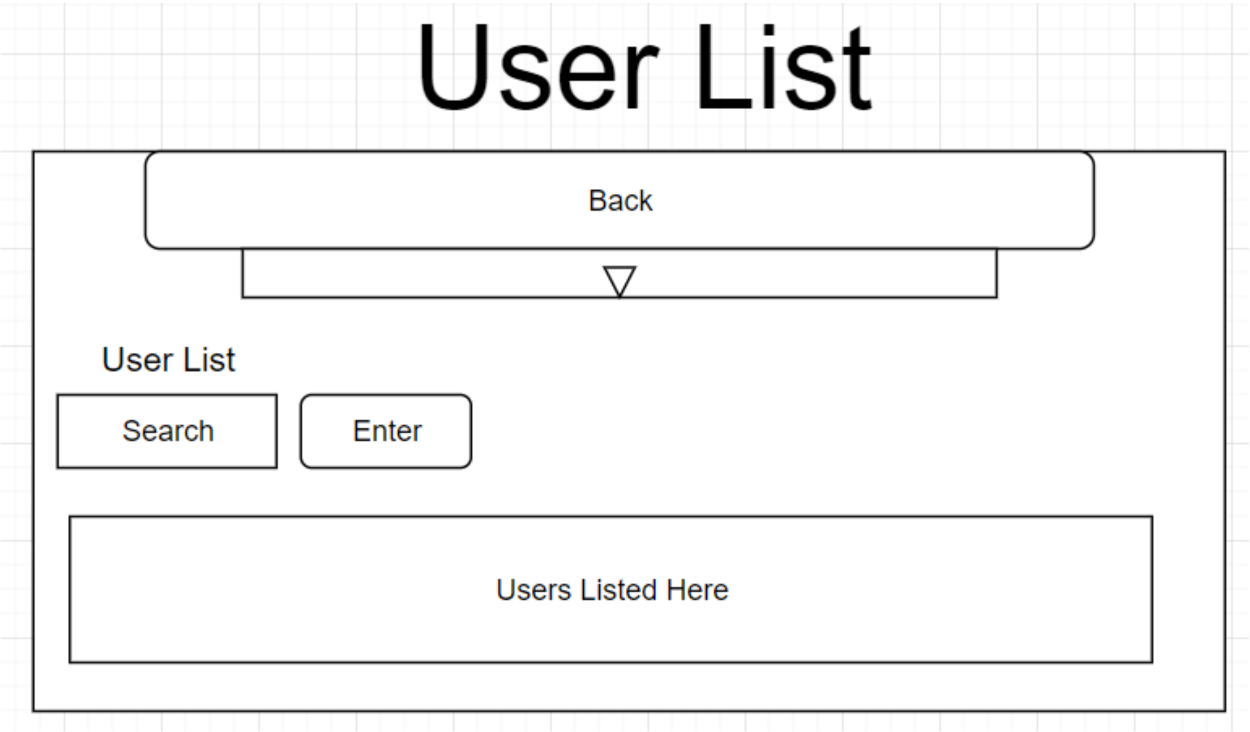

Reset Password
Back
▽
Reset Password:
Email:
Submit

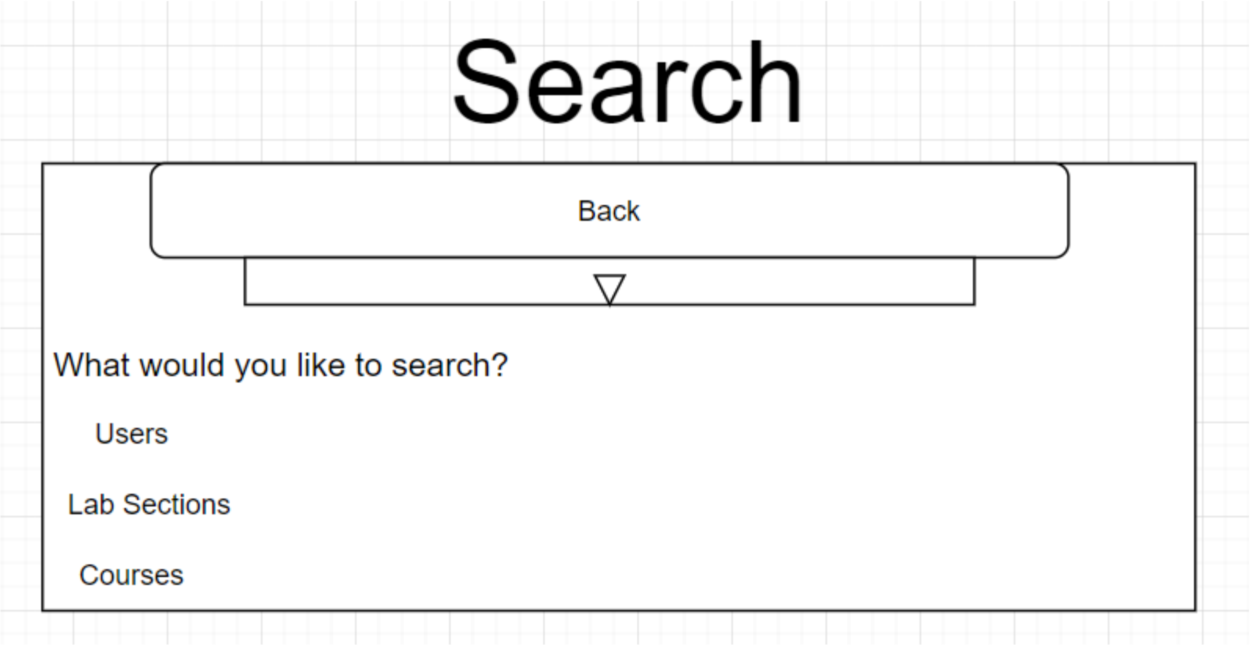
Course List



Lab Section List



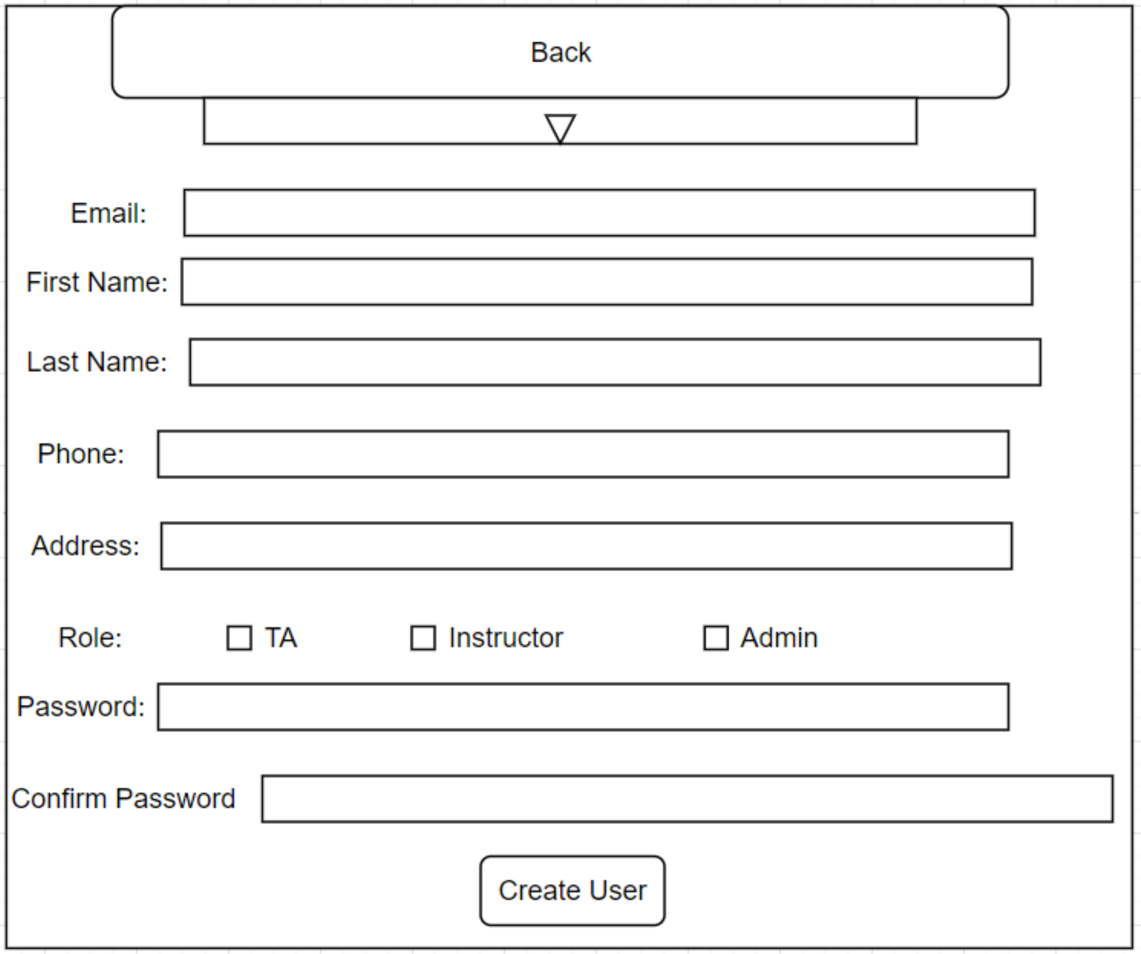

User List
Back
▽
User List
Search
Enter
Users Listed Here



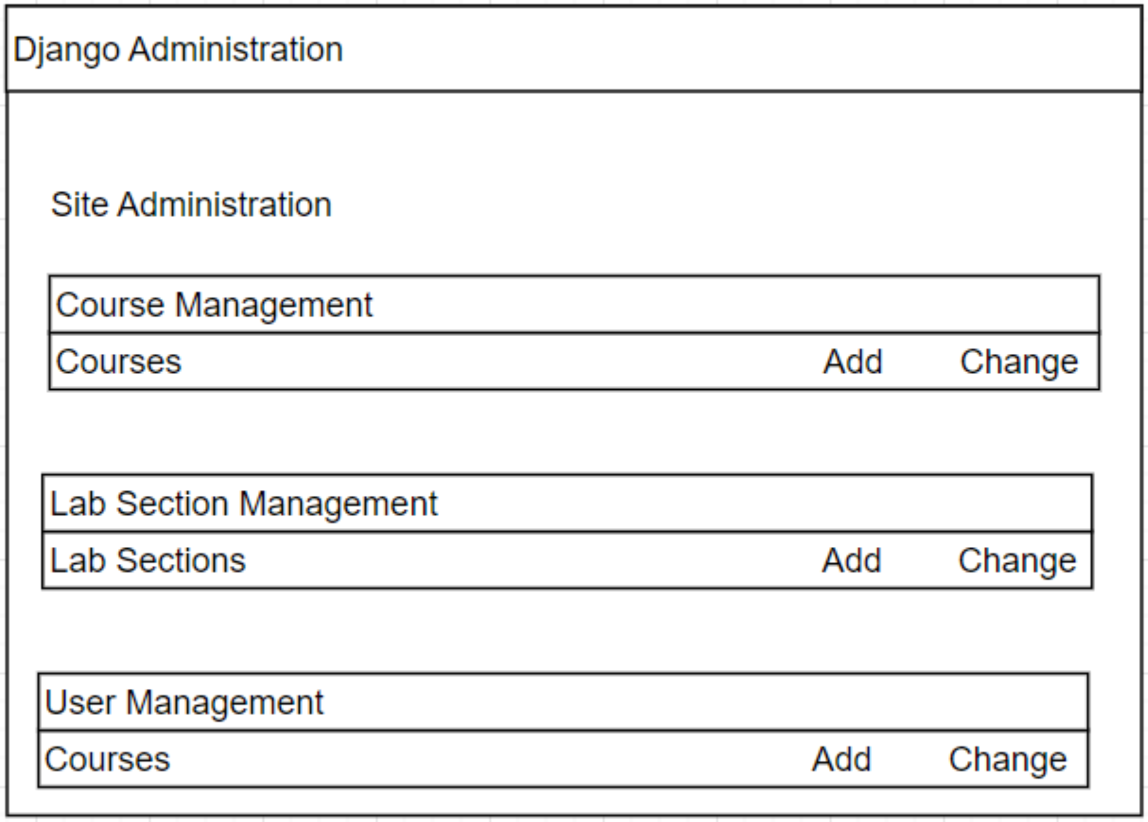
Search

What would you like to search?

Create User


Back
▽
Email:
First Name:
Last Name:
Phone:
Address:
Role:
□ TA
□ Instructor
□ Admin
Password:
Confirm Password
Create User

Database Page



Django Administration

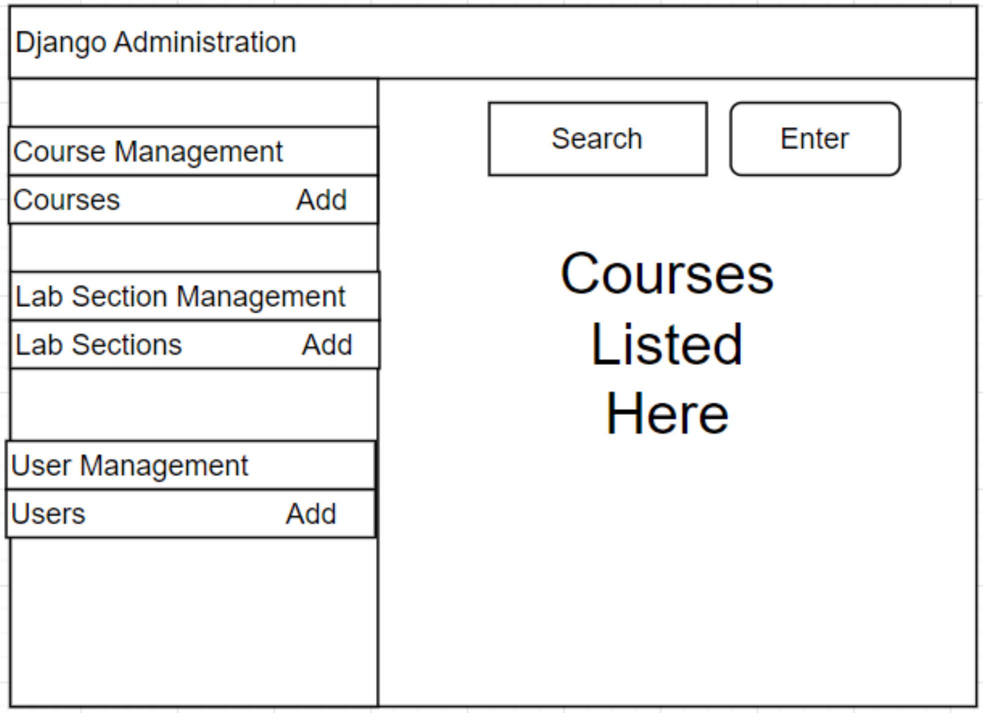
Site Administration

Course Management		
Courses	Add	Change

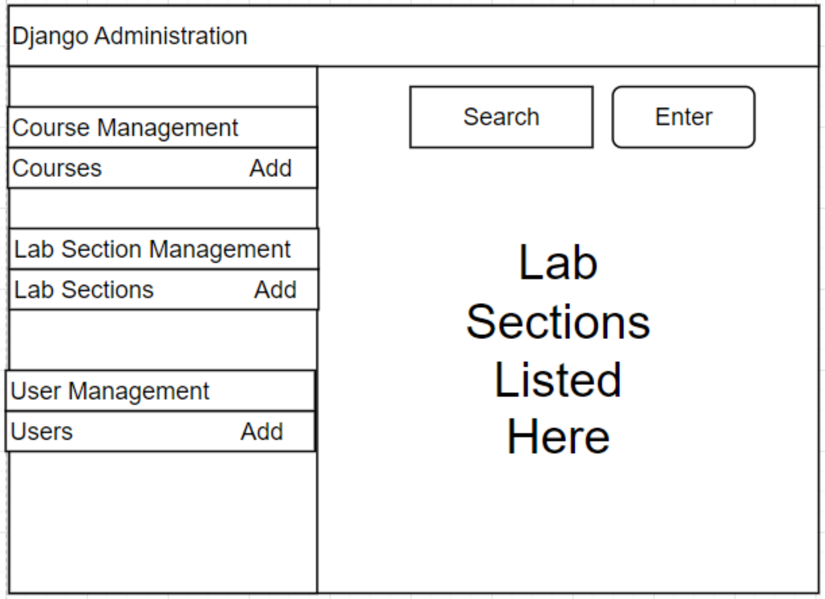
Lab Section Management		
Lab Sections	Add	Change

User Management		
Courses	Add	Change

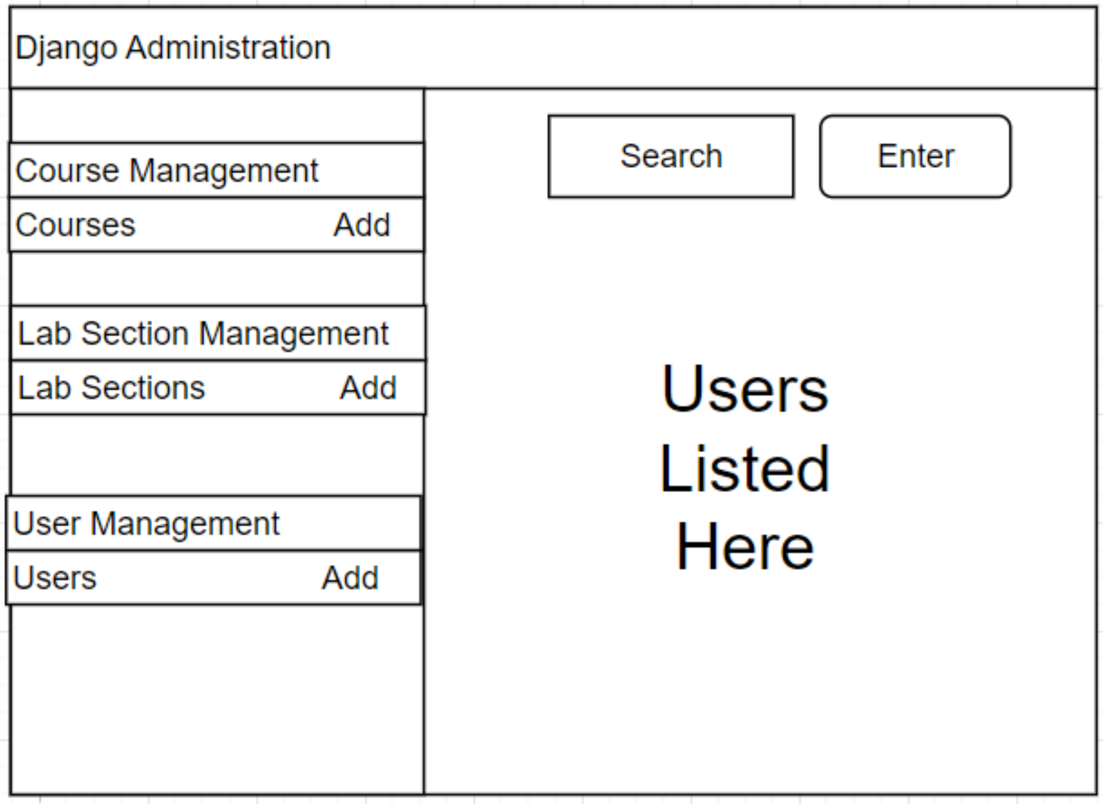
Course Database Page



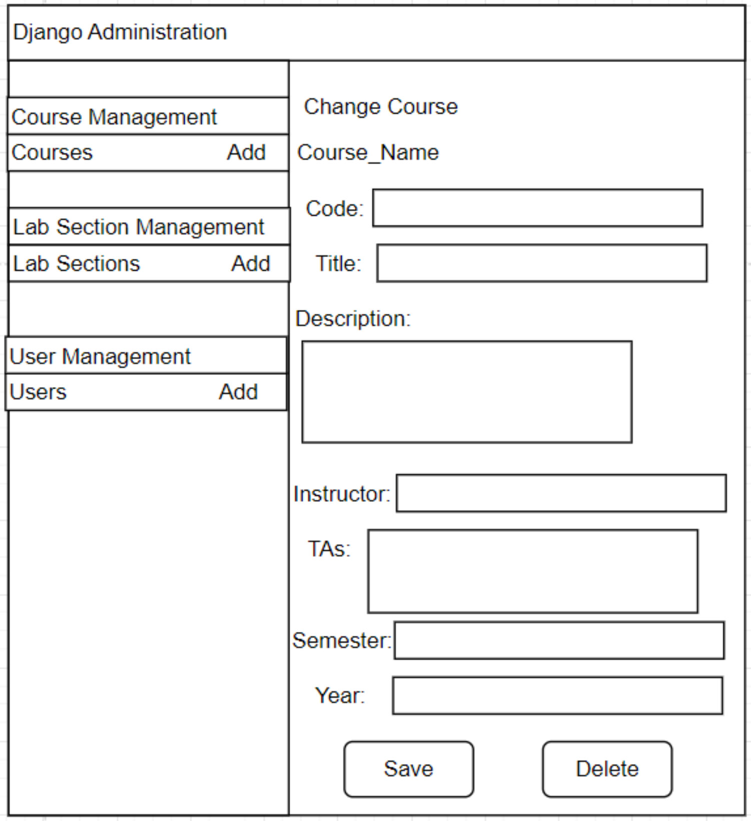
Lab Section Database Page



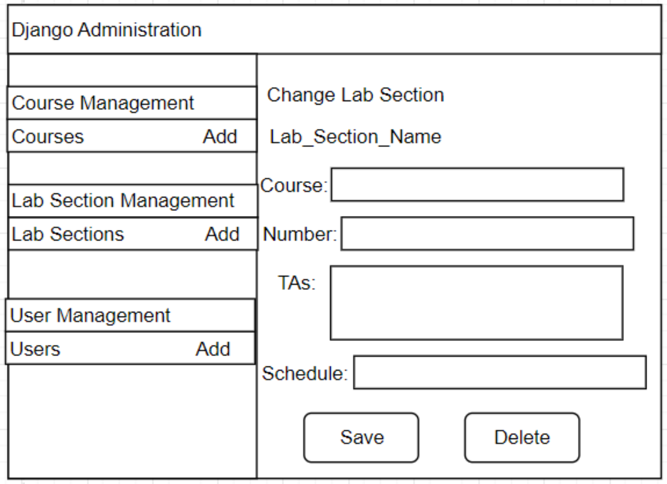
User Database Page



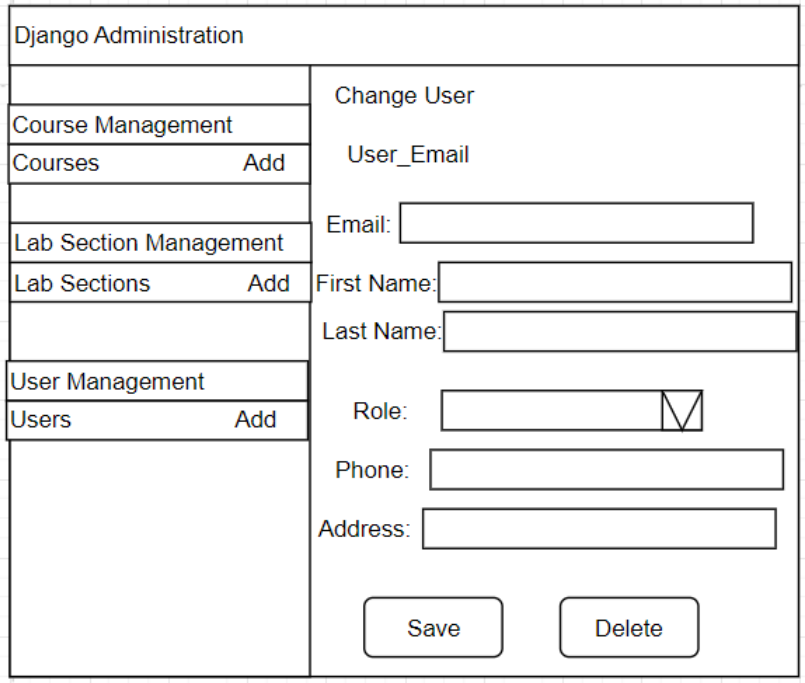
Modify Course Database Page



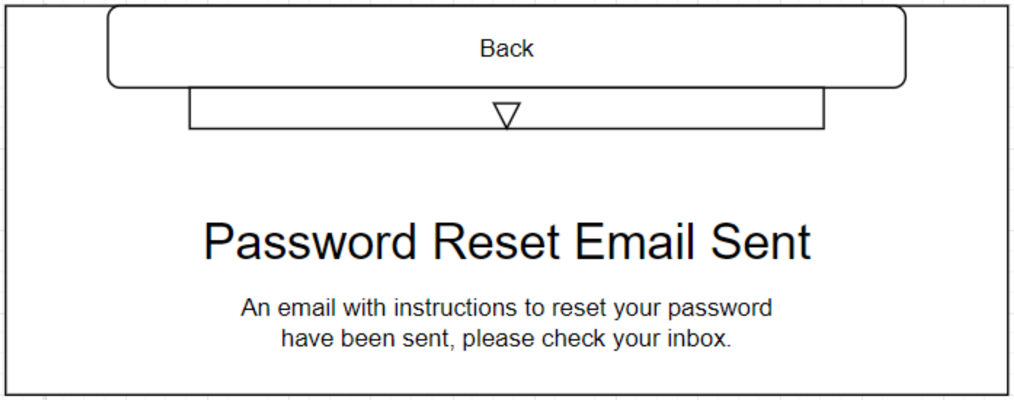
Modify Lab Section Database Page



Modify User Database Page



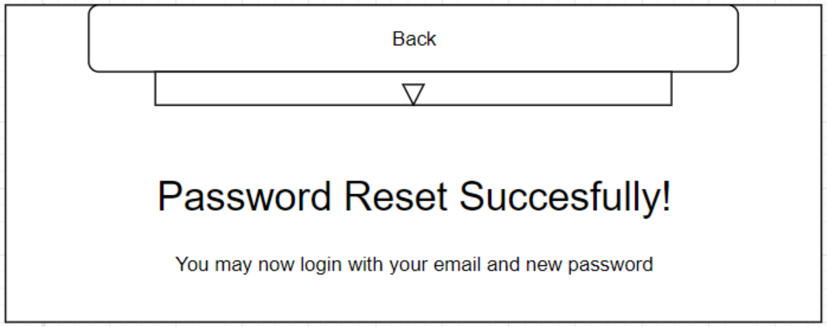
Password Email Sent Page



Password Reset Email Sent

An email with instructions to reset your password
have been sent, please check your inbox.

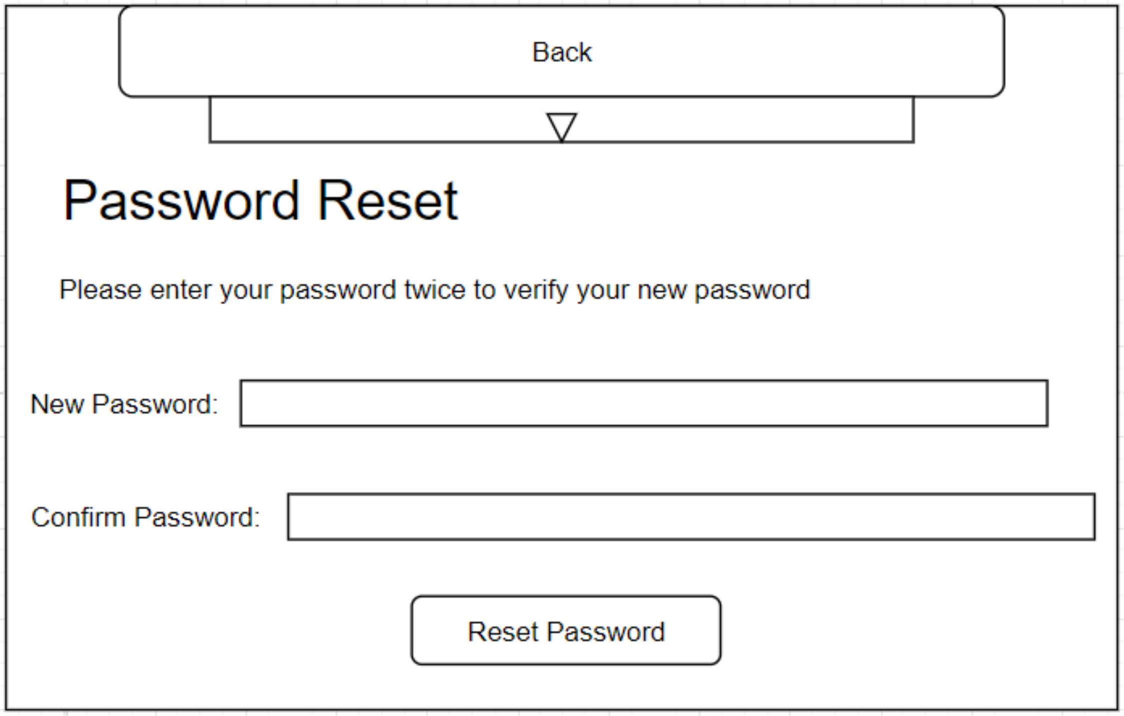
Password Succesfully Reset Page



Password Reset Succesfully!

You may now login with your email and new password

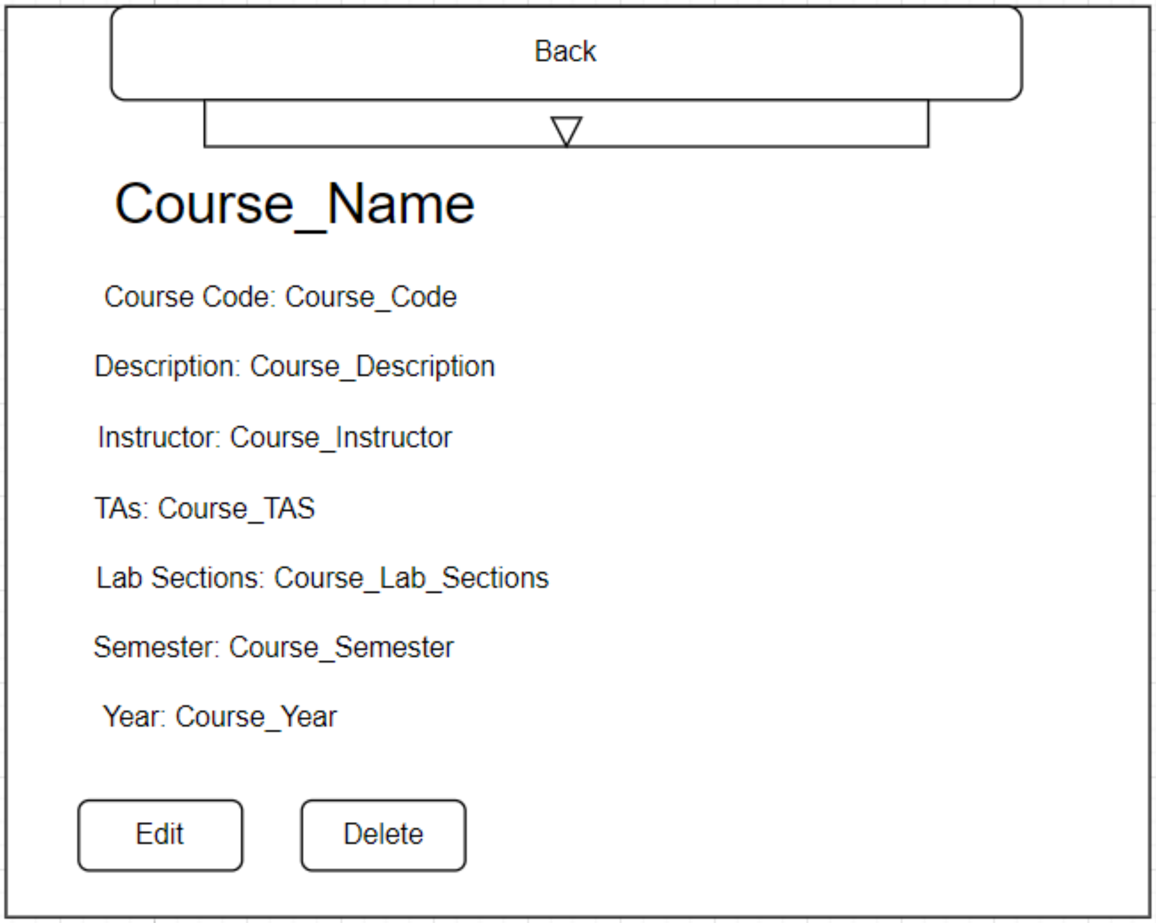
Password Reset Page



Password Reset

Please enter your password twice to verify your new password

Display Course Page



Course_Name

Course Code: Course_Code

Description: Course_Description

Instructor: Course_Instructor

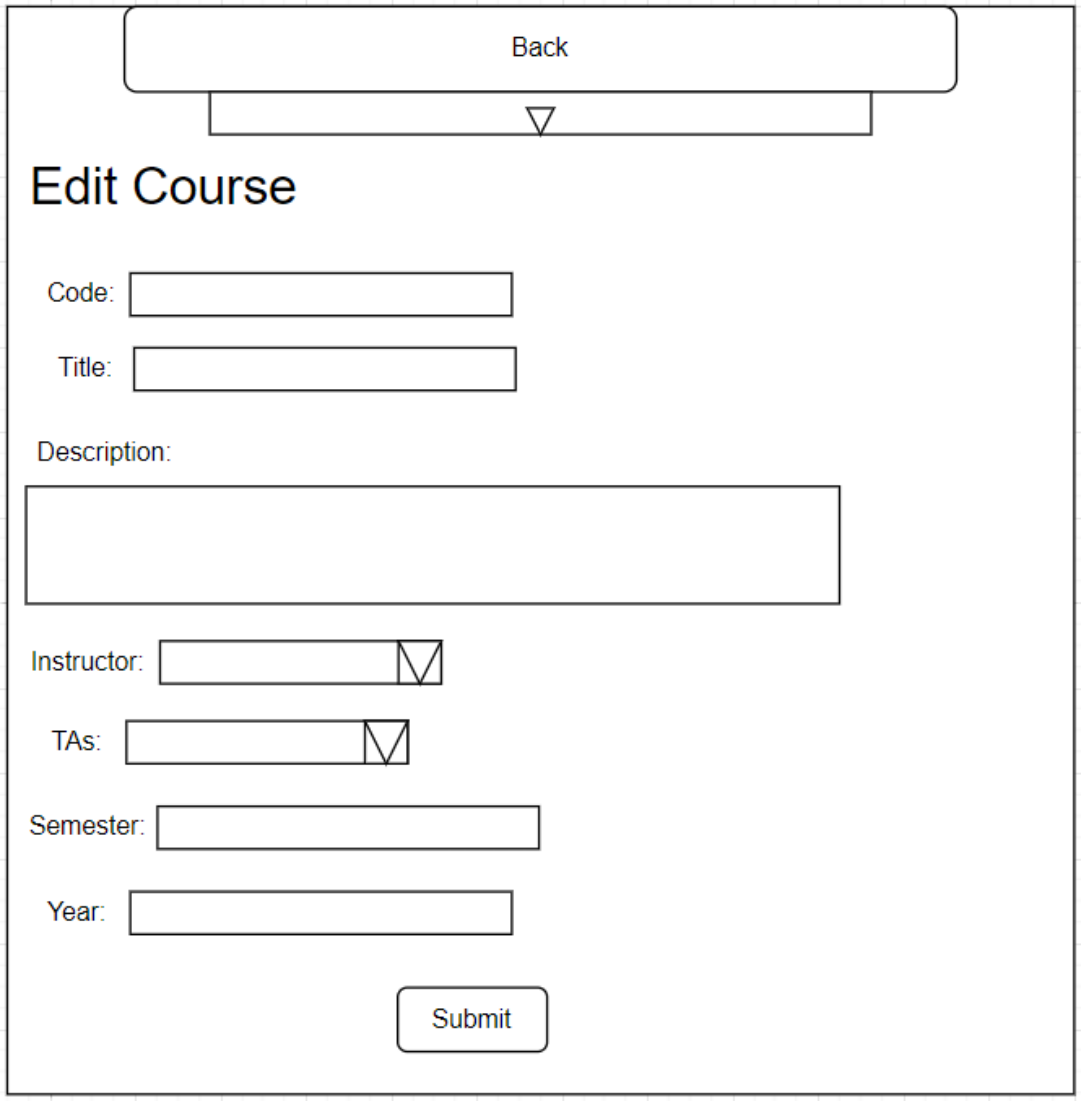
TAs: Course_TAS

Lab Sections: Course_Lab_Sections

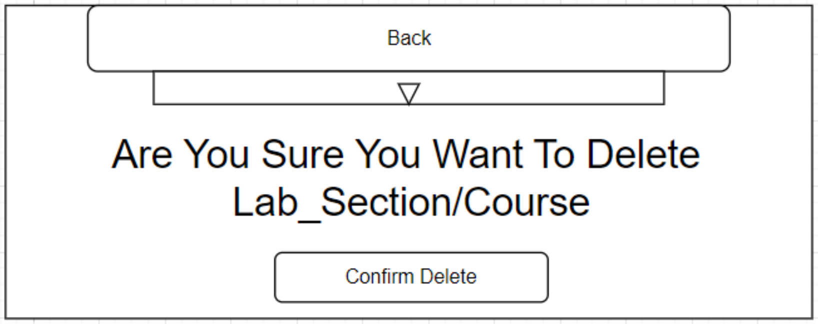
Semester: Course_Semester

Year: Course_Year

Edit Course Page

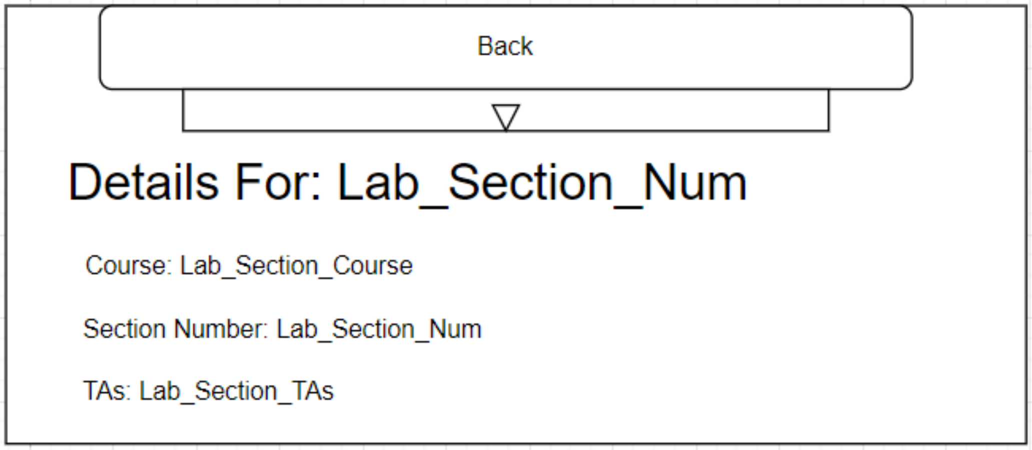


Delete Lab Section & Course Page



Are You Sure You Want To Delete Lab_Section/Course

Display Lab Section Page



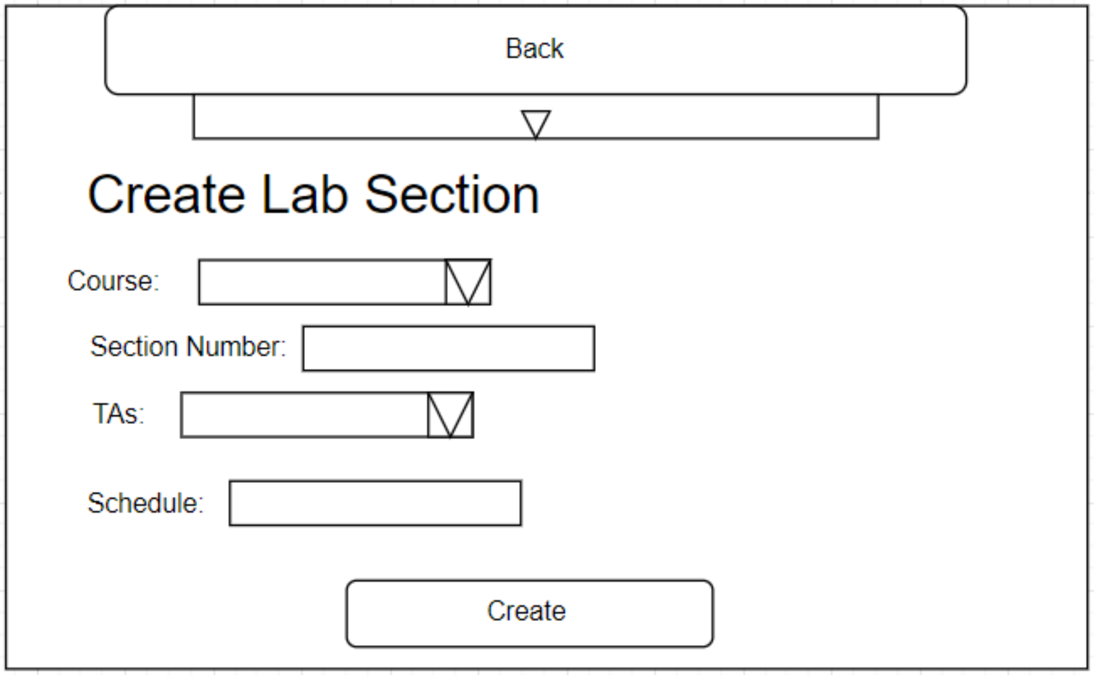
Details For: Lab_Section_Num

Course: Lab_Section_Course

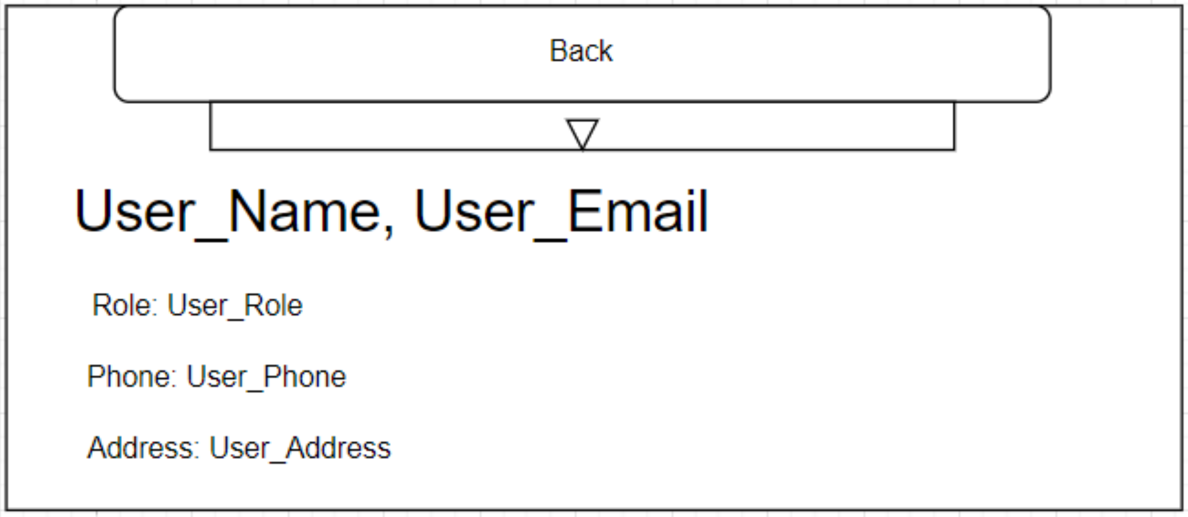
Section Number: Lab_Section_Num

TAs: Lab_Section_TAs

Create Lab Section Page



Display User Page



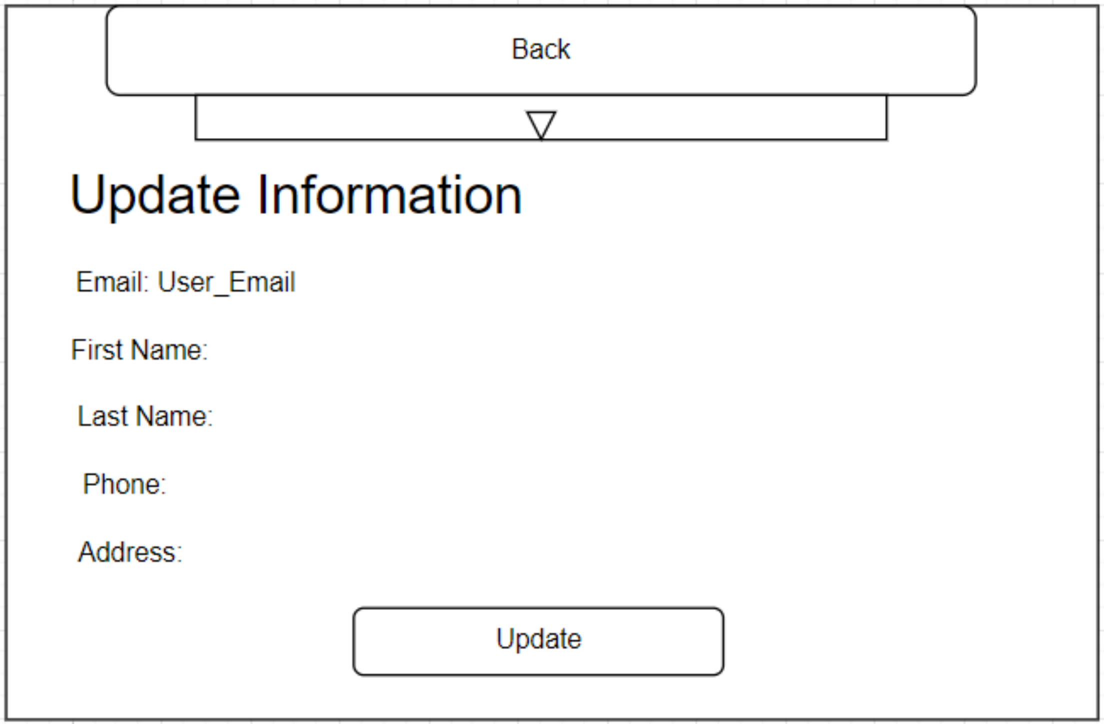
User_Name, User_Email

Role: User_Role

Phone: User_Phone

Address: User_Address

Update Information Page



Update Information

Email: User_Email

First Name:

Last Name:

Phone:

Address:

PBIs:

1. As a supervisor/administrator, I want to be able to create a new accounts for instructors or TAs.

Time Estimate: M

Acceptance Criteria:

Scenario: Instructor Account Creation - GIVEN an account with the role of the administrator AND within the admin webpage WHEN I click on the 'create account' button THEN I specify user, password, and e-mail AND I specify the role of 'Instructor.'

Scenario: TA Account Creation - GIVEN an account with the role of the administrator AND within the admin webpage WHEN I click on the 'create account' button THEN I specify user, password, and e-mail AND I specify the role of 'Instructor.'

Scenario: Primary Key Overlap - GIVEN the supervisor submits an existing email within the database

Number of Acceptance Tests: 3

2. As a supervisor, I want the ability to create, delete, and edit accounts for users in the system, ensuring accurate and current information.

Time Estimate: L

Acceptance Criteria:

Scenario: Supervisor creates a new user account. GIVEN the supervisor has logged into the system. WHEN the supervisor starts the creation of a new user account. THEN the system should ask the supervisor for user account details

Scenario: Supervisor provides valid information to create a new user account.   GIVEN the supervisor is asked for user account details. WHEN the supervisor provides necessary information for a new account. THEN the system should accept the provided information

Scenario: Supervisor provides existing user information for a new user account. GIVEN the supervisor is asked for user account details. WHEN the supervisor provides information for an account that already exists. THEN the system should show an error message. AND ask the supervisor to provide unique information

Number of Acceptance Tests: 2

3. As a supervisor, I want to be able to create new courses in the system with specific details such as course code, name, and semester

Time Estimate: M

Acceptance Criteria:

Scenario: Supervisor creates a new course GIVEN the supervisor has logged into the system WHEN the supervisor tries to create a new course THEN the system should ask the supervisor for the course's details

Scenario: Supervisor provides valid information for the course they want to create. GIVEN the supervisor is asked for the course's details WHEN the supervisor submits a unique course code, a course name, and specifies the semester THEN the system should accept the provided information AND the system should confirm the course was created successfully

Scenario: Supervisor provides a course code that already exists. GIVEN the supervisor is asked for course details WHEN the supervisor provides a course code that already exists in the system THEN the system should show an error message AND ask the supervisor to provide a unique course code

Scenario: Supervisor provides incomplete course information. GIVEN the supervisor is asked for course details WHEN the supervisor provides incomplete information or misses a required field THEN the system should show an error message AND guide the supervisor to where the error is occurring

Scenario: Supervisor views the list of created courses. GIVEN the supervisor has successfully created a new course WHEN the supervisor navigates to view the list of courses THEN the system should display the newly created course in the list

Scenario: Course notification system. GIVEN the supervisor has successfully created a new course WHEN the supervisor sends notifications about the new course THEN the notification system should receive and send the notification via UWM email

Scenario: Unauthorized access to course creation. GIVEN a user without supervisor/administrator access WHEN attempting to creating a course THEN the system should deny access AND show an unauthorized access message

Number of Acceptance Tests: 3

4. As a user, I want to be able to log-in.

Time Estimate: S

Acceptance Criteria:

Scenario - Instructor Log-In - GIVEN an Instructor enters a valid username and password WHEN the instructor clicks submit THEN they will be directed to the instructor home page

Scenario - Invalid Log-in - GIVEN an user enters an invalid username or password. WHEN the user clicks the submit button. THEN an error message will be displayed

Scenario - Admin Log-In - GIVEN an Admin enters a valid username and password. WHEN the admin click the submit button. THEN they will be directed to the admin home page

Scenario - TA Log-In - GIVEN a TA enters a valid username and password. WHEN the TA clicks the submit button. THEN they will be redirected to the TA home page

Number of Acceptance Tests: 2