**Lab Manuals**

**Software Quality Assurance Technology**

****

**SUBMITTED BY:**

**Muhammad Khubaib**

**ROLL NO:**

**23-SET-017**

**SEMESTR:**

**4th**

**SUBMITTED TO:**

**Hafiz Muhammad Umar Hayat**

**DEPARTMENT OF SOFTWARE ENGINEERING TECHNOLOGY**

# Contents

**2**

* 1. [Abstraction](#_bookmark1)

[**1 Lab 1 - Functional and Non-Functional Requirements**](#_bookmark0)

. 2

. . .

* 1. [Non-Functional Requirements](#_bookmark3)

. . . . . . . . . . . .

[1.4.1 Simple](#_bookmark5) . . . . . . . . .

[1.4 Log files](#_bookmark4)

[1.2 Functional Requirements](#_bookmark2)

. 2

. 2

. 2

. 2

### 3

[**2 Lab 2 - Use Case Diagram**](#_bookmark6)

### 3

. 3

. . .

* 1. [Entity Relation Diagram](#_bookmark10)
  2. [Database Diagram](#_bookmark9)
  3. [Entities](#_bookmark8)

[**3 Lab 3 - ERD and Database Diagram**](#_bookmark7)

. 3

. 3

### 3

[**4 Lab 4 - Creating the relational Database**](#_bookmark11)

### 5

[**5 Lab 5 - Creating a Data Access Layer for recipe Manipulation**](#_bookmark12)

### 11

. 11

[**6 Lab 6 - Create Interfaces**](#_bookmark13)

. 11

. 12

. 13

### 13

[8.2 Validation](#_bookmark21)

[8.1 Verification](#_bookmark20)

[**7 Lab 7 - Business Logic**](#_bookmark18)

|  |  |  |
| --- | --- | --- |
| [6.1 Add Recipe](#_bookmark14) | | .  . |
| [6.2 Edit Recipe](#_bookmark15) | |
| [6.3 Delete Recipe](#_bookmark16) | | |
| [6.4 Search](#_bookmark17) | . . . . | |

### 16

. 16

[**8 Lab 8 - Verification and Validation**](#_bookmark19)

. 17

|  |  |  |
| --- | --- | --- |
| [8.2.1 Getting the recipe](#_bookmark22) | |  |
| [8.2.2 Uploading the recipe](#_bookmark23) | | |
| [8.2.3 Editing the recipe](#_bookmark24) | | . |
| [8.2.4 Delete the recipe](#_bookmark25) | . . | |

. 17

. 18

. 18

. 20

### 21

[**9 Lab 9 - Recipe Presentation Module**](#_bookmark26)

### 24

[**10 Lab 10 - Payment Module**](#_bookmark27)

### 25

[12.1 Integrating Modules](#_bookmark30)

.

[12.1.2 Admin Login](#_bookmark32)

[12.1.1 User Login](#_bookmark31)

[**12 Lab 12 - Integration**](#_bookmark29)

[**11 Lab 11 - Admin Panel**](#_bookmark28)

### 26

. 26

. 26

. 28

# Lab 1 - Functional and Non-Functional Re- quirements

## Abstraction

The website is to be a simple place to share and discover new receipies where users can upload new receipies. Having an admin along with a bunch of users. The users would be able to see others receipies and upload their receipies in markdown format. The admin will have the power to remove a recipy and to block users.

## Functional Requirements

* As an user, I shall be able to sign-up.
* As a user or admin, I shall be able to log-in.
* As a user, I shall be able to upload a new recipe.
* As a user, I shall be able to edit mine uploaded recipe.
* As a user, I shall be able to delete mine uploaded recipe.
* As an admin, I shall be able to delete any receipe.
* As an admin, I shall be able to block any user.

## Non-Functional Requirements

* Fast
* Reliable
* Scalable
* Modular
* User Friendly

## Log files

* + 1. **Simple**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SR NO | FR NO | Title | Priority | Date | Submited By | Status | Remarks |
| 1 | 1 | user sign-up | high | *¡2025-01-23 Thu¿* | Wahab | Approved | - |
| 2 | 2 | user/admin login | high | *¡2025-01-23 Thu¿* | Wahab | Approved | - |
| 3 | 3 | upload recipe | high | *¡2025-01-23 Thu¿* | Wahab | Approved | - |
| 4 | 4 | edit recipies | low | *¡2025-01-23 Thu¿* | Wahab | Approved | - |
| 5 | 5 | delete recipe | high | *¡2025-01-23 Thu¿* | Wahab | Approved | - |
| 6 | 6 | admin delete | high | *¡2025-01-23 Thu¿* | Wahab | Approved | - |
| 7 | 7 | admin block | low | *¡2025-01-23 Thu¿* | Wahab | Approved | - |

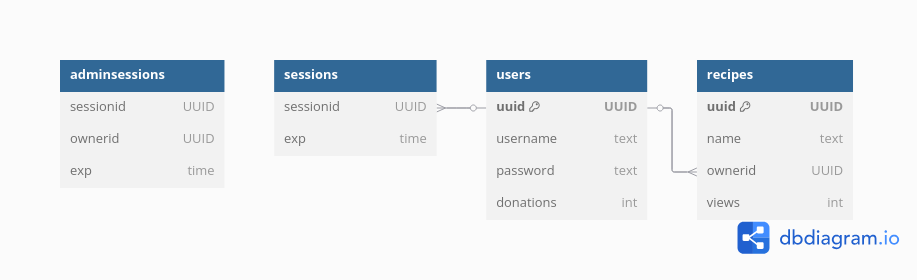
# Lab 2 - Use Case Diagram

Below is a use case diagram of Recipedia

# Lab 3 - ERD and Database Diagram

## Entities

* Use
* Admin
  1. **Database Diagram**

****

* 1. **Entity Relation Diagram**

****

# Lab 4 - Creating the relational Database

We create the relation db by using the following query,

BEGIN;

CREATE TABLE IF NOT EXISTS public.adminsessions (

sessionid text COLLATE pg\_catalog."default" NOT NULL,

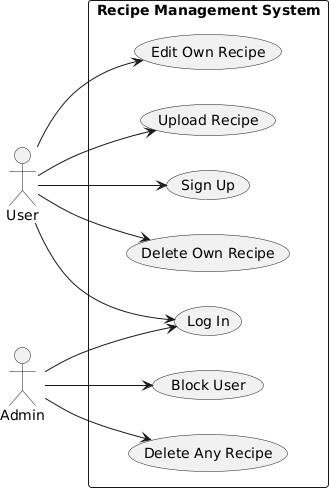


Figure 1: use case diagram

exp text COLLATE pg\_catalog."default" NOT NULL, CONSTRAINT adminsessions\_pkey PRIMARY KEY (sessionid)

);

CREATE TABLE IF NOT EXISTS public.recipes (

uuid uuid NOT NULL,

name text COLLATE pg\_catalog."default" NOT NULL, ownerid uuid NOT NULL,

views integer,

CONSTRAINT recipes\_pkey PRIMARY KEY (uuid)

);

CREATE TABLE IF NOT EXISTS public.sessions (

sessionid text COLLATE pg\_catalog."default" NOT NULL, ownerid text COLLATE pg\_catalog."default",

exp text COLLATE pg\_catalog."default", CONSTRAINT uuid PRIMARY KEY (sessionid)

);

CREATE TABLE IF NOT EXISTS public.users (

uuid text COLLATE pg\_catalog."default" NOT NULL, name text COLLATE pg\_catalog."default",

password text COLLATE pg\_catalog."default", CONSTRAINT "primary" PRIMARY KEY (uuid)

);

ALTER TABLE IF EXISTS public.sessions

ADD CONSTRAINT "session-ownerid" FOREIGN KEY (ownerid) REFERENCES public.users (uuid) MATCH SIMPLE

ON UPDATE NO ACTION ON DELETE NO ACTION NOT VALID;

END;

# Lab 5 - Creating a Data Access Layer for recipe Manipulation

Following is the golang code for ’DAL’ for Recipe table,

package database

import ( "big/internal/modals" "database/sql"

)

func (s \*service) AddRecipe(recipe \*modals.Recipe) error { q := ‘

INSERT INTO recipes(uuid, name, ownerid, views) VALUES($1, $2, $3, -1)

‘

\_, err := s.db.Exec(q, recipe.UUID, recipe.Name, recipe.OwnerId) if err != nil {

return err

}

return nil

}

func (s \*service) GetRecipe(UUID string) (\*modals.Recipe, error) { var recipe modals.Recipe

q := ‘

SELECT \* FROM recipes WHERE uuid = $1;

‘

row := s.db.QueryRow(q, UUID)

err := row.Scan(&recipe.UUID, &recipe.Name, &recipe.OwnerId, &recipe.Views) if err != nil {

if err == sql.ErrNoRows { return nil, ErrItemNotFound

} else {

return nil, err

}

}

return &recipe, nil

}

func (s \*service) DeleteRecipe(uuid string) error { q := "DELETE FROM recipes WHERE uuid = $1"

res, err := s.db.Exec(q, uuid) if err != nil {

return err

}

rowsAffected, err := res.RowsAffected() if err != nil {

return err

}

if rowsAffected <= 0 { return ErrItemNotFound

}

return nil

}

func (s \*service) DeleteRecipeByUser(userUUid string) error { q := "DELETE FROM recipes WHERE ownerid = $1"

\_, err := s.db.Exec(q, userUUid) if err != nil {

return err

}

return nil

}

func (s \*service) MostViewedRecipes() ([]modals.Recipe, error) { var recipes []modals.Recipe

rows, err := s.db.Query("SELECT \* FROM recipes ORDER BY views LIMIT 10;") if err != nil {

return nil, err

}

defer rows.Close()

for rows.Next() {

var recipe modals.Recipe

err := rows.Scan(&recipe.UUID, &recipe.Name, &recipe.OwnerId, &recipe.Views) if err != nil {

return nil, err

}

recipes = append(recipes, recipe)

}

return recipes, nil

}

func (s \*service) SearchRecipe(name string) ([]modals.Recipe, error) { var recipes []modals.Recipe

searchTerm := "%" + name + "%"

query := "SELECT \* FROM recipes WHERE name ILIKE $1"

rows, err := s.db.Query(query, searchTerm) if err != nil {

return nil, err

}

defer rows.Close()

for rows.Next() {

var recipe modals.Recipe

err := rows.Scan(&recipe.UUID, &recipe.Name, &recipe.OwnerId, &recipe.Views) if err != nil {

return nil, err

}

recipes = append(recipes, recipe)

}

return recipes, nil

}

func (s \*service) IncreaseRecipeViews(recipe \*modals.Recipe) error { q := ‘

UPDATE recipes

SET views = views + 1 WHERE uuid = $1

‘

res, err := s.db.Exec(q, recipe.UUID) if err != nil {

return err

}

rowsAffected, err := res.RowsAffected() if err != nil {

return err

}

if rowsAffected <= 0 { return ErrItemNotFound

}

return nil

}

func (s \*service) EditRecipeName(uuid string, name string) error { q := ‘

UPDATE recipes SET name = $1 WHERE uuid = $2 ‘

res, err := s.db.Exec(q, name, uuid) if err != nil {

return err

}

rowsAffected, err := res.RowsAffected() if err != nil {

return err

}

if rowsAffected <= 0 { return ErrItemNotFound

}

return nil

}

func (s \*service) GetRecipesByUser(name string) ([]modals.Recipe, error) { var recipes []modals.Recipe

user, err := s.GetUserByName(name) if err != nil {

return nil, err

}

q := "SELECT \* FROM recipes WHERE ownerid = $1" rows, err := s.db.Query(q, user.UUID)

if err != nil { return nil, err

}

for rows.Next() {

var recipe modals.Recipe

err := rows.Scan(&recipe.UUID, &recipe.Name, &recipe.OwnerId, &recipe.Views) if err != nil {

return nil, err

}

recipes = append(recipes, recipe)

}

return recipes, nil

}

func (s \*service) NumberOfRecipes() int { var numberOfRecipes int

q := ‘

SELECT COUNT(\*) FROM recipes; ‘

s.db.QueryRow(q).Scan(&numberOfRecipes)

return numberOfRecipes

}

func (s \*service) GetAllRecipes() ([]modals.Recipe, error) { var recipes []modals.Recipe

q := ‘SELECT \* FROM recipes;‘ rows, err := s.db.Query(q)

if err != nil { return nil, err

}

defer rows.Close()

for rows.Next() {

var recipe modals.Recipe

err := rows.Scan(&recipe.UUID, &recipe.Name, &recipe.OwnerId, &recipe.Views) // Adjust fi if err != nil {

return nil, err

}

recipes = append(recipes, recipe)

}

if err := rows.Err(); err != nil { return nil, err

}

return recipes, nil

}

# Lab 6 - Create Interfaces

Following are the interfaces for Recipe Management defined in htmx

## Add Recipe

{{ define "head" }}

<title> Upload Recipe </title>

{{ end }}

{{ define "body" }}

<h1> Upload a Recipe </h1>

<div class="form-group">

<form action="/api/add-recipe" method="post">

<div>

<p>

<label for="Name">Name</label>

<br>

<input type="text" name="name">

</p>

</div>

<div>

<p>

<label for="Content">Write here</label>

<br>

<textarea type="text" name="content" rows="10" cols="10"></textarea>

</p>

</div>

<div>

<p>

<button type="submit">Submit</button>

</p>

</div>

</form>

</div>

{{ end }}

## Edit Recipe

{{ define "head" }}

<title>Edit Recipe</title>

{{ end }}

{{ define "body" }}

<h1> Edit {{ .ViewModel.RecipeName }} </h1>

<form action="/api/edit-recipe/{{ .ViewModel.UUID }}" method="post">

<div>

<p>

<label for="Name">Name</label>

<br>

<input type="text" name="name" value="{{ .ViewModel.RecipeName }}">

</p>

</div>

<div>

<p>

<label for="Content">Write here</label>

<br>

<textarea type="text" name="content" rows="10" cols="10">{{ .ViewModel.RecipeContent

</p>

</div>

<div>

<p>

<button type="submit">Submit</button>

</p>

</div>

</form>

{{ end }}

## Delete Recipe

{{ define "userCards" }}

<div class="recipe-container">

{{ range .ViewModel.Recipes }}

<div class="recipe-card">

<h3 class="recipe-name">{{ .Name }}</h3>

<p class="recipe-views">Views: {{ .Views }}</p>

<button class="option-button" onclick="window.location.href=‘/view/edit/{{ .UUID }}‘" Edit

</button>

<button class="delete-button" onclick="window.location.href=‘/api/delete/{{ .UUID }}‘;"

Delete

</button>

</div>

{{ end }}

</div>

{{ end }}

## Search

<form class="search-bar" action="/view/search" >

<input class="search-input" name="search" type="text" placeholder="Search" aria-label="

<button class="search-button">Search</button>

</form>

# Lab 7 - Business Logic

Following is the business logic for Recipe Module

func (api \*api) UploadRecipe(w http.ResponseWriter, r \*http.Request) {

var recipeInfo RecipeInfo

err := json.NewDecoder(r.Body).Decode(&recipeInfo)

if err != nil {

fmt.Printf("Can’t get recipe Info cause, %s", err) http.Error(w, "Invalid Request", http.StatusBadRequest)

}

// Get session token

c, err := r.Cookie("session-token") if err != nil {

fmt.Println("Can’t find Cookie")

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

// Get user session db := api.db

session, err := db.GetSession(c.Value) if err != nil {

fmt.Printf("Can’t find Session %s, cause %s\n", c.Value, err.Error()) http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

recipe := modals.NewRecipe(recipeInfo.Name, session.OwnerId) err = db.AddRecipe(recipe)

if err != nil {

fmt.Printf("Can’t Add the recipes cause, %s\n", err)

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

// Create directory for recipe

directoryPath := "upload/recipes/" + recipe.UUID if api.fileExists(directoryPath) { fmt.Println("Recipe Directory Already Exists")

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

err = api.fs.Mkdir(directoryPath, 0755) if err != nil {

fmt.Printf("Error creating directory: %s\n", err.Error()) http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

// Creating an html file

err = api.mdFileGenreator(recipeInfo.Content, recipe.UUID) if err != nil {

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

// Creating an html file

err = api.htmlFileGenerator(recipe.Name, recipeInfo.Content, recipe.UUID) if err != nil {

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

w.WriteHeader(http.StatusOK)

}

func (api \*api) EditRecipeHandler(w http.ResponseWriter, r \*http.Request) { vars := mux.Vars(r)

recipeUUID := vars["id"] var recipeInfo RecipeInfo

err := json.NewDecoder(r.Body).Decode(&recipeInfo) if err != nil {

fmt.Printf("Can’t parse Json cause %s \n", err) http.Error(w, "Bad JSON provided", http.StatusBadRequest) return

}

err = api.deleteRecipeFiles(recipeUUID) if err != nil {

fmt.Printf("Can’t delete filese cause, %s \n", err)

http.Error(w, "Can’t delete recipe files", http.StatusInternalServerError) return

}

err = api.mdFileGenreator(recipeInfo.Content, recipeUUID) if err != nil {

fmt.Printf("Can’t generate md cause %s \n", err)

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

err = api.htmlFileGenerator(recipeInfo.Name, recipeInfo.Content, recipeUUID) if err != nil {

fmt.Printf("Can’t generate html cause, %s \n", err)

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

db := api.db

err = db.EditRecipeName(recipeUUID, recipeInfo.Name) fmt.Printf("Provided recipe name, %s", recipeInfo.Name) if err != nil {

fmt.Printf("Can’t change recipe name cause, %s \n", err)

http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

w.WriteHeader(http.StatusOK)

}

func (api \*api) DeleteRecipeHandler(w http.ResponseWriter, r \*http.Request) { vars := mux.Vars(r)

recipeUUID := vars["id"]

directoryPath := "upload/recipes/" + recipeUUID

if !(api.authSameUser(r)) {

fmt.Println("Doesn’t have the permission to edit recipe")

http.Error(w, "Doesn’t have the permission to edit recipe", http.StatusInternalServerErro return

}

db := api.db

err := db.DeleteRecipe(recipeUUID) if err != nil {

fmt.Printf("Can’t find Recipe To Delete cause, %s\n", err.Error()) http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

if !api.fileExists(directoryPath) {

fmt.Printf("Can’t find recipe directory to delete in path, %s\n", directoryPath) http.Error(w, "Internal Server Error", http.StatusInternalServerError)

return

}

err = os.RemoveAll(directoryPath) if err != nil {

fmt.Printf("Can’t find Delete recipe cause, %s\n", err.Error()) http.Error(w, "Internal Server Error", http.StatusInternalServerError) return

}

w.WriteHeader(http.StatusOK)

}

# Lab 8 - Verification and Validation

## Verification

Here is the verification that the api provided performs all the requirements for ’Recipe Module’ We have the following endpoints,

func (api \*api) UploadRecipe(w http.ResponseWriter, r \*http.Request)

func (api \*api) EditRecipeHandler(w http.ResponseWriter, r \*http.Request) func (api \*api) DeleteRecipeHandler(w http.ResponseWriter, r \*http.Request) func (api \*api) RecipeMdContent(w http.ResponseWriter, r \*http.Request)

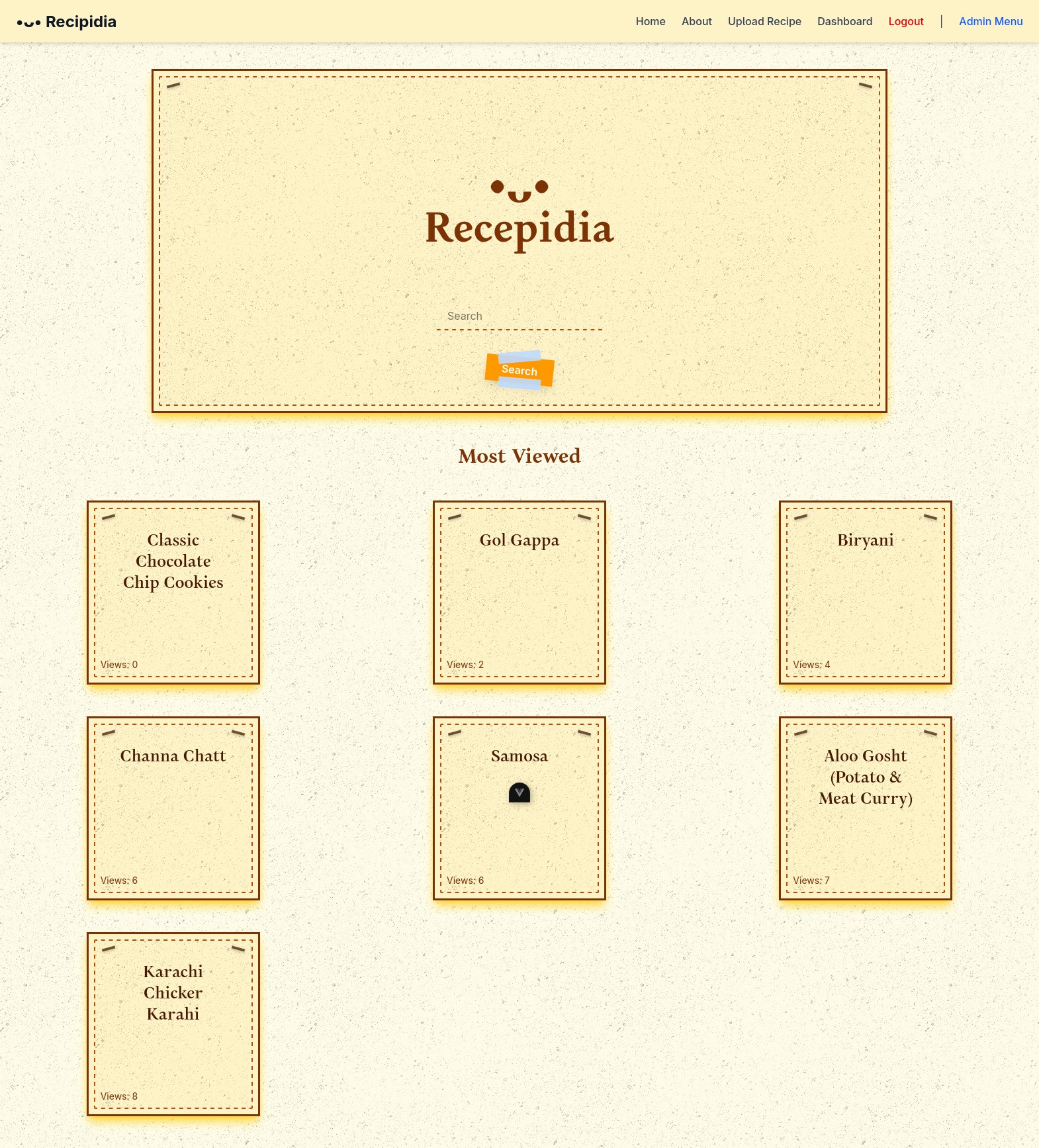
to statisfy the following functional requirements

* As a user, I shall be able to upload a new recipe.
* As a user, I shall be able to edit mine uploaded recipe.
* As a user, I shall be able to delete mine uploaded recipe. to statisfy the following functional requirements

## Validation

### Getting the recipe

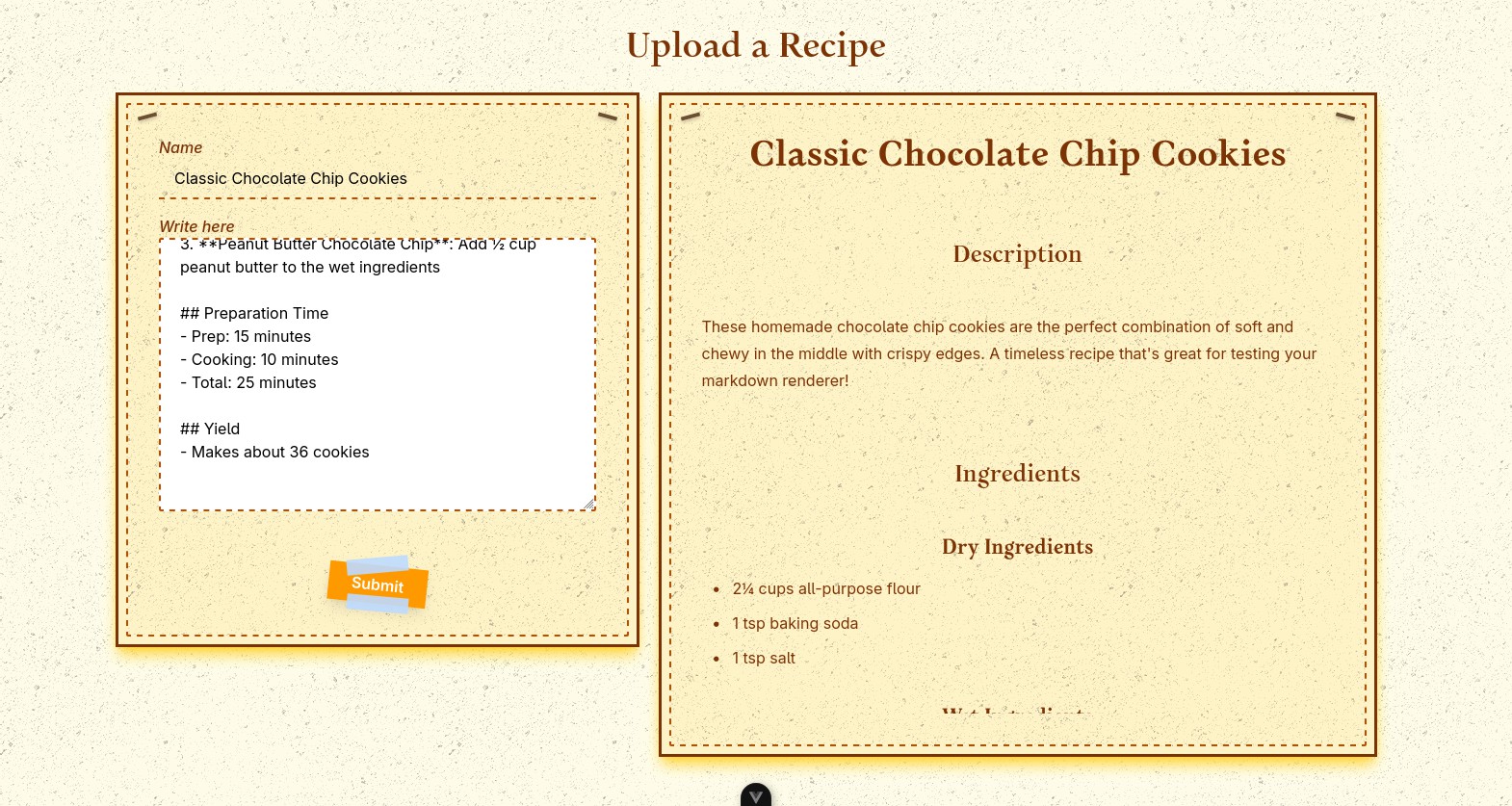
* + - 1. Go to Homepage.



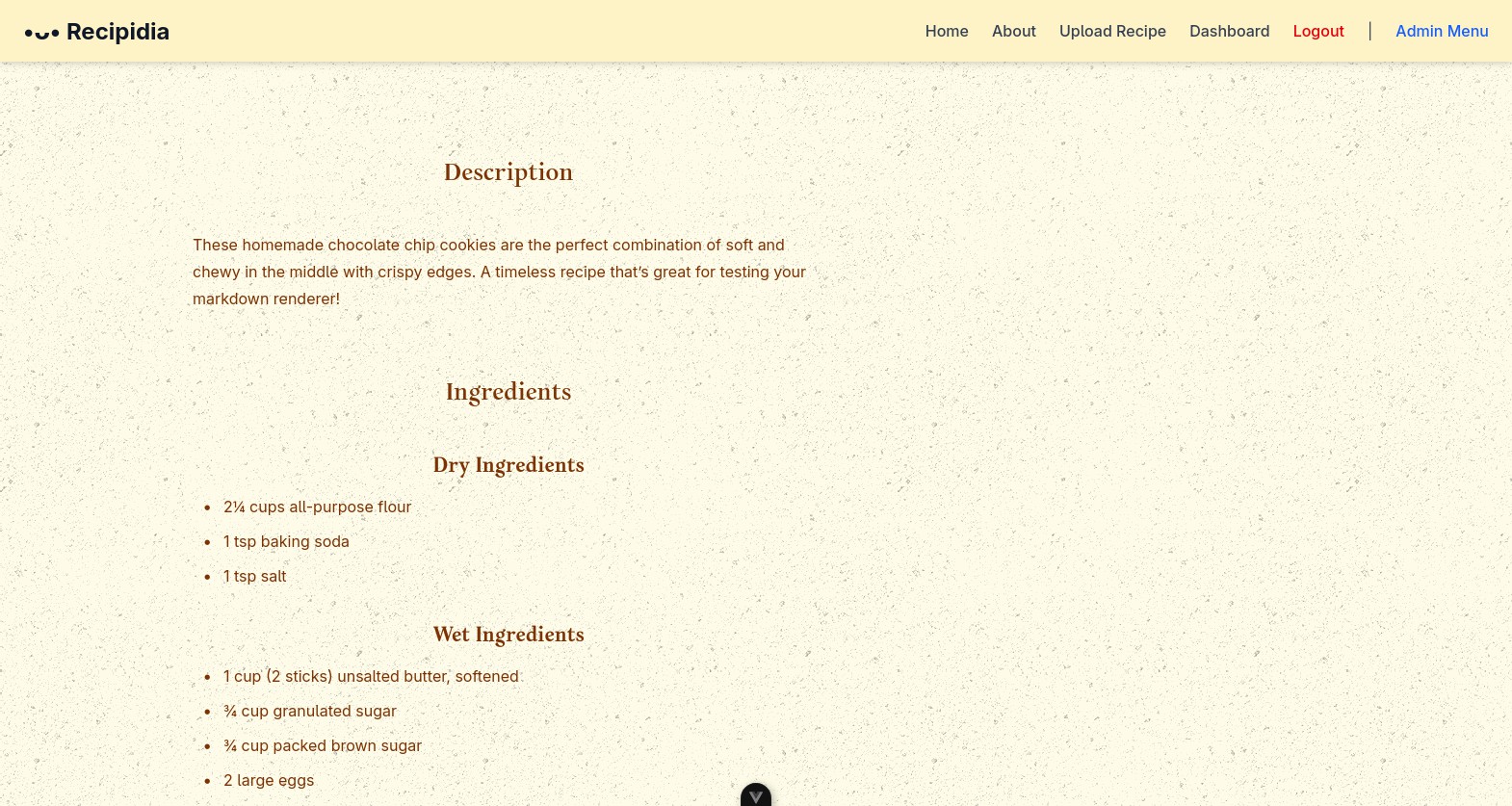
1. Click on any recipe.

### Uploading the recipe

* + - 1. Writting the recipe.

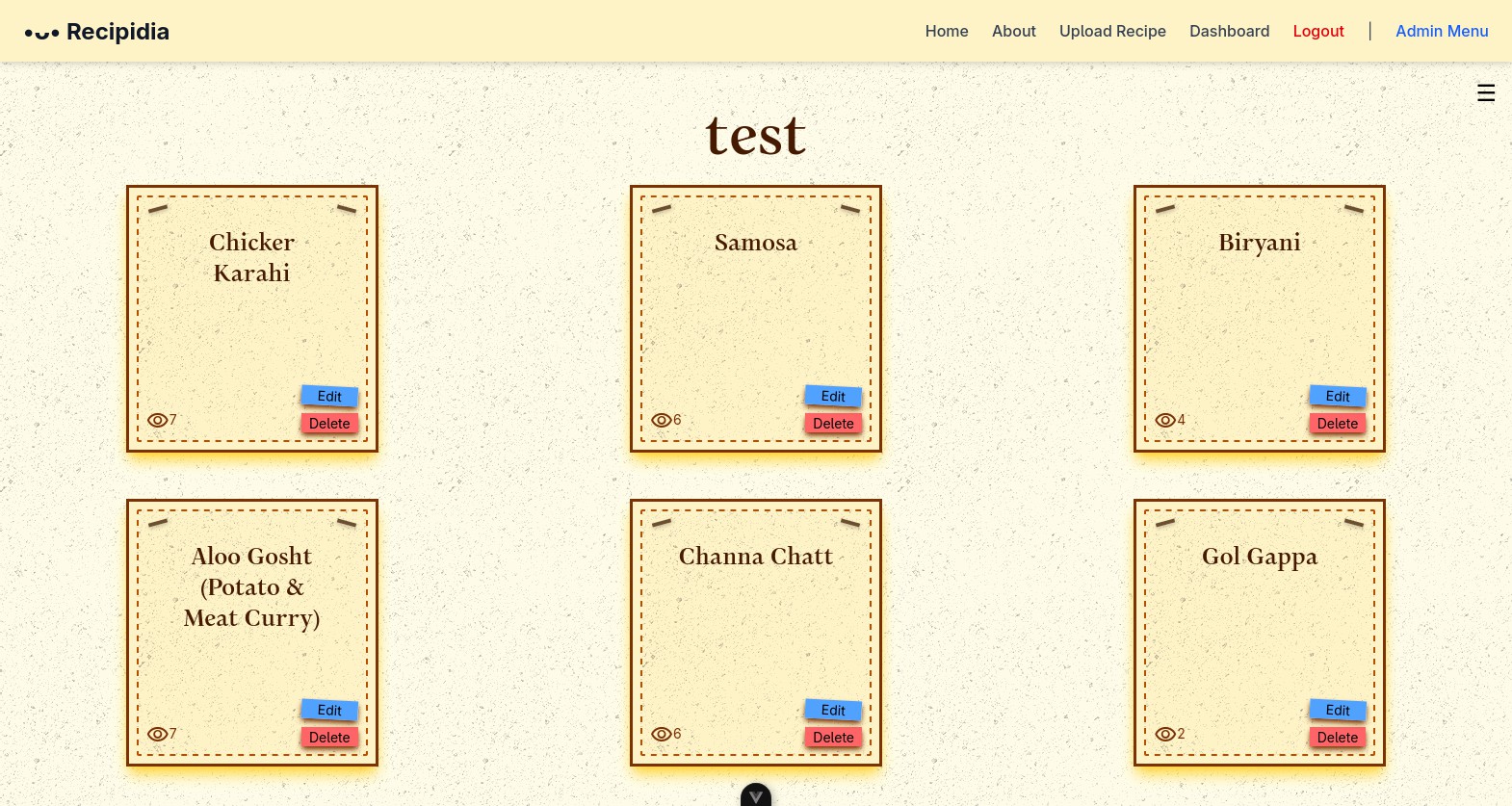


1. Click the submit button. And the submitted recipe will open.



### Editing the recipe

* + - 1. Go to dashboard.



1. Click the edit button on the recipe you want to edit.

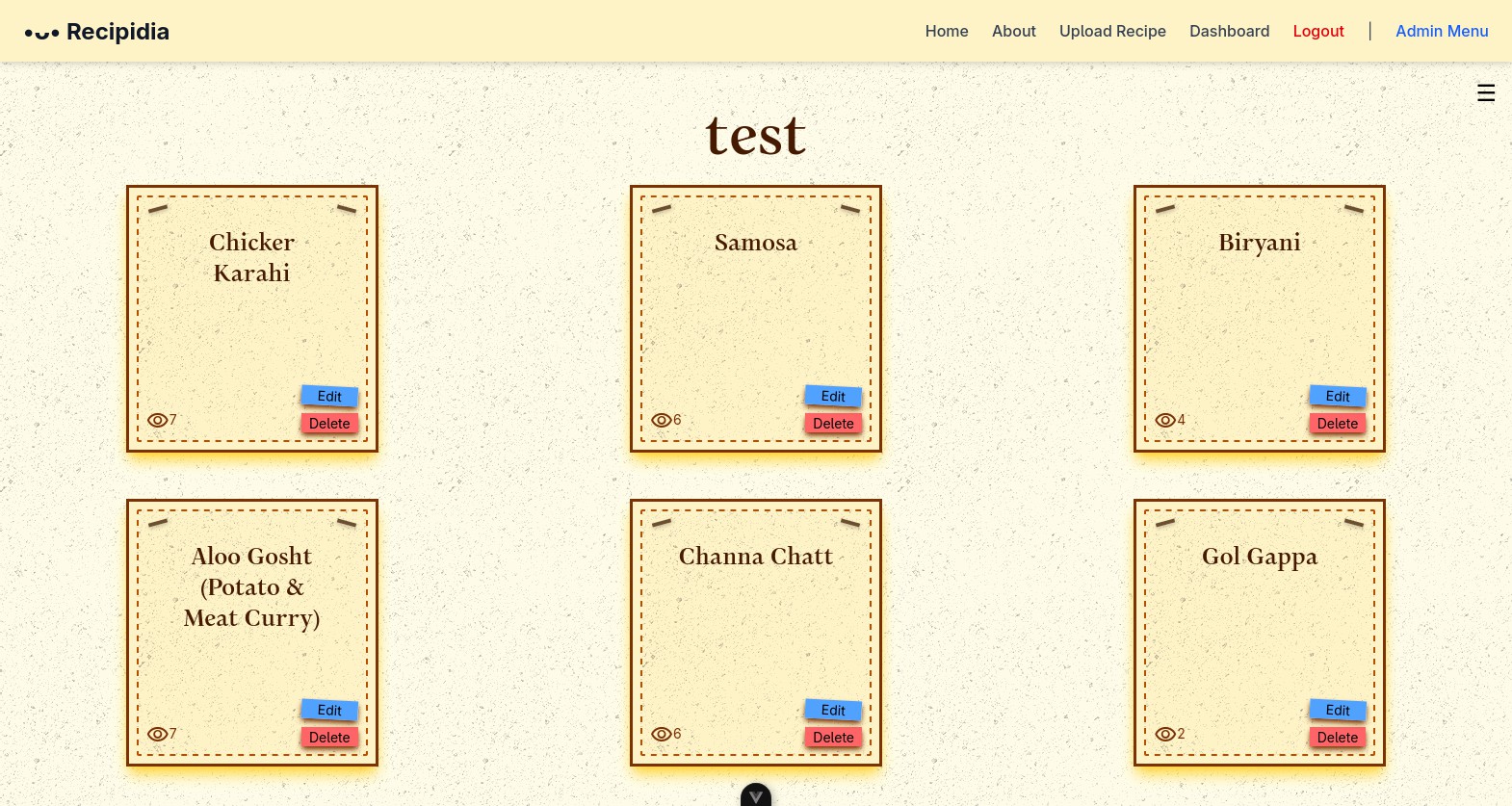


1. Click the submit button. The submitted recipe will open.



### Delete the recipe

* + - 1. Go to dashboard.



1. Click the delete button of the recipe you want.

# Lab 9 - Recipe Presentation Module

Following is the Business Logic for ’Search Recipe, Top Recipe Listing and View Recipe and Getting Total Number of Recipes’

func (api \*api) SearchRecipeHandler(w http.ResponseWriter, r \*http.Request) { searchterm := r.URL.Query().Get("searchTerm")

fmt.Printf("\n%s\n", searchterm) db := api.db

recipes, err := db.SearchRecipe(searchterm) if err != nil {

http.Error(w, "No recipes found", http.StatusInternalServerError) fmt.Printf("Can’t get the searched recipes, %s", err.Error()) return

}

jsonData, err := json.Marshal(recipes) if err != nil {

http.Error(w, "Can’t Marshall json", http.StatusInternalServerError)

}

w.Header().Set("Content-Type", "application/json") w.WriteHeader(http.StatusOK)

w.Write(jsonData)

}

func (api \*api) ServeRecipe(w http.ResponseWriter, r \*http.Request) { vars := mux.Vars(r)

db := api.db

recipe, err := db.GetRecipe(vars["id"]) if err != nil {

http.Error(w, "Can’t Render the Recipe", http.StatusInternalServerError) fmt.Printf("Can’t get the most viewed recipes cause, %s", err.Error()) return

}

directoryPath := "upload/recipes/" + recipe.UUID + "/recipe.html" if !api.fileExists(directoryPath) {

http.Error(w, "Can’t find this recipe", http.StatusInternalServerError) fmt.Printf("Can’t get the most viewed recipes cause, %s", recipe.UUID) return

}

// Wrap Afero FS as io/fs.FS fsWrapper := afero.NewIOFS(api.fs)

tmpl, err := template.ParseFS(fsWrapper, directoryPath) err = tmpl.Execute(w, tmpl)

if err != nil {

http.Error(w, "Can’t find this recipe", http.StatusInternalServerError) fmt.Printf("Can’t get the most viewed recipes cause, %s", recipe.UUID) return

}

err = db.IncreaseRecipeViews(recipe) if err != nil {

http.Error(w, "Can’t increase recipe’s views", http.StatusInternalServerError) fmt.Printf("Can’t increase recipe’s views cause, %s", err.Error())

return

}

}

func (api \*api) RecipeInfoHandler(w http.ResponseWriter, r \*http.Request) { vars := mux.Vars(r)

db := api.db

recipe, err := db.GetRecipe(vars["id"]) if err != nil {

http.Error(w, "Can’t Render the Recipe", http.StatusInternalServerError) fmt.Printf("Can’t get the most viewed recipes cause, %s", err.Error()) return

}

filePath := "upload/recipes/" + recipe.UUID + "/recipe.md" if !api.fileExists(filePath) {

http.Error(w, "Can’t find this recipe", http.StatusInternalServerError) fmt.Printf("\nCan’t get the most viewed recipes cause, %s\n", recipe.UUID) return

}

jsonData, err := json.Marshal(recipe.Name) if err != nil {

http.Error(w, ‘{error: "Failed to fetch recipes"}‘, http.StatusInternalServerError) return

}

w.Header().Set("Content-Type", "application/json") w.WriteHeader(http.StatusOK)

w.Write(jsonData)

}

func (api \*api) RecipeMdContent(w http.ResponseWriter, r \*http.Request) { vars := mux.Vars(r)

db := api.db

recipe, err := db.GetRecipe(vars["id"]) if err != nil {

http.Error(w, "Can’t Render the Recipe", http.StatusInternalServerError) fmt.Printf("Can’t get the most viewed recipes cause, %s", err.Error()) return

}

fmt.Printf("Serving Recipe %s", recipe.Name)

directoryPath := "upload/recipes/" + recipe.UUID + "/recipe.md"

if !api.fileExists(directoryPath) {

http.Error(w, "Can’t find this recipe", http.StatusInternalServerError) fmt.Printf("Can’t get the most viewed recipes cause, %s", recipe.UUID) return

}

content, err := os.ReadFile(directoryPath) mdContent := string(content)

w.Header().Set("Content-Type", "text/plain") w.WriteHeader(http.StatusOK) w.Write([]byte(mdContent))

}

func (api \*api) MostViewedRecipesHandler(w http.ResponseWriter, r \*http.Request) { recipes, err := api.db.MostViewedRecipes()

if err != nil {

http.Error(w, "Can’t get any recipes", http.StatusNotFound)

}

jsonData, err := json.Marshal(recipes) if err != nil {

http.Error(w, "Can’t Marshall json", http.StatusInternalServerError)

}

w.Header().Set("Content-Type", "application/json") w.WriteHeader(http.StatusOK)

w.Write(jsonData)

}

# Lab 10 - Payment Module

package api import ( "net/http"

"github.com/gorilla/mux"

)

func (api \*api) DonateRecipeHandler(w http.ResponseWriter, r \*http.Request) { vars := mux.Vars(r)

recipeID := vars["id"] amountStr := vars["amount"]

recipe, err := api.db.GetRecipe(recipeID) if err != nil {

http.Error(w, "Recipe not found", http.StatusNotFound) return

}

owner, err := api.db.GetUserByUUid(recipe.OwnerId) if err != nil {

http.Error(w, "Owner not found", http.StatusNotFound) return

}

owner.IncreaseDonation(amountStr) w.WriteHeader(http.StatusOK) w.Write([]byte("Donation recorded"))

}

# Lab 11 - Admin Panel

Following is the Business Logic for Admin Panel writen in htmx

{{ define "head" }}

<title> Dashboard </title>

{{ end }}

{{ define "body" }}

<h1>Hello Sir Mr Admin</h1>

<h2>Dashboard</h2>

<div class="info-container">

<div class="info-card">

<h3 class="info-name">Number of Users</h3>

<p class="info">{{ .ViewModel.NOU }}</p>

</div>

<div class="info-card">

<h3 class="info-name">Number of Recipes</h3>

<p class="info">{{ .ViewModel.NOR }}</p>

</div>

</div>

{{ end }}

and the business logic behind

func (api \*api) AdminDashboardDataHandler(w http.ResponseWriter, r \*http.Request) {

db := api.db

numberOfRecipes := db.NumberOfRecipes() numberOfUsers := db.NumberOfUsers()

data := struct {

NumberOfRecipes int ‘json:"numberOfRecipes"‘ NumberOfUsers int ‘json:"numberOfUsers"‘

}{

NumberOfRecipes: numberOfRecipes, NumberOfUsers: numberOfUsers,

}

jsonData, err := json.Marshal(data)

if err != nil {

http.Error(w, "Internal Database Error", http.StatusInternalServerError)

}

w.Header().Set("Content-Type", "application/json") w.WriteHeader(http.StatusOK)

w.Write(jsonData)

}

# Lab 12 - Integration

## Integrating Modules

Following is the working of login module working in integration with ’user’ and ’customer’ module.

### User Login

Following is the api handler for user login

func (api \*api) LoginHandler(w http.ResponseWriter, r \*http.Request) {

var userReq UserRequest

err := json.NewDecoder(r.Body).Decode(&userReq)

if err != nil {

http.Error(w, "Invalid JSON input", http.StatusBadRequest) fmt.Printf("Can’t login cause, %s \n", err.Error())

}

fmt.Printf("\nSession Cookie, name: %s\n", userReq.Name) db := api.db

user, err := db.GetUserByName(userReq.Name) // Gets user from user module if err != nil {

http.Error(w, "Incorrect Password", http.StatusInternalServerError) fmt.Printf("\nCan’t find user cause, %s\n", err)

return

}

if !user.CheckPassword(userReq.Password) { // Checking Passwords http.Error(w, "Incorrect Password", http.StatusInternalServerError) fmt.Printf("Password Incorrect\n")

return

}

session := api.createCookie(w, user.UUID) err = db.AddSession(session)

if err != nil {

http.Error(w, "Database Error", http.StatusInternalServerError) fmt.Printf("Can’t add the session to db cause, %s\n", err.Error()) return

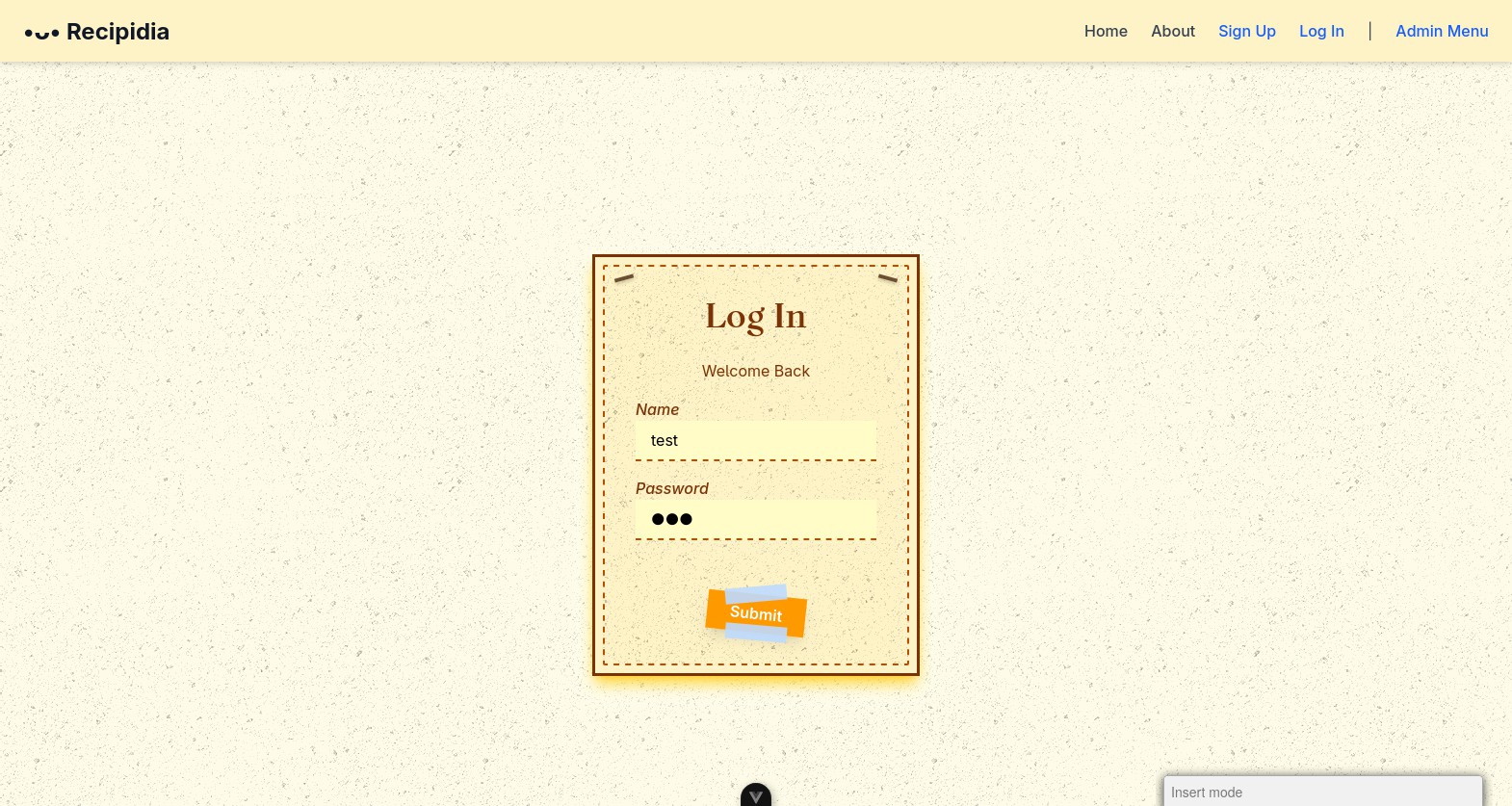
}

w.WriteHeader(http.StatusOK)

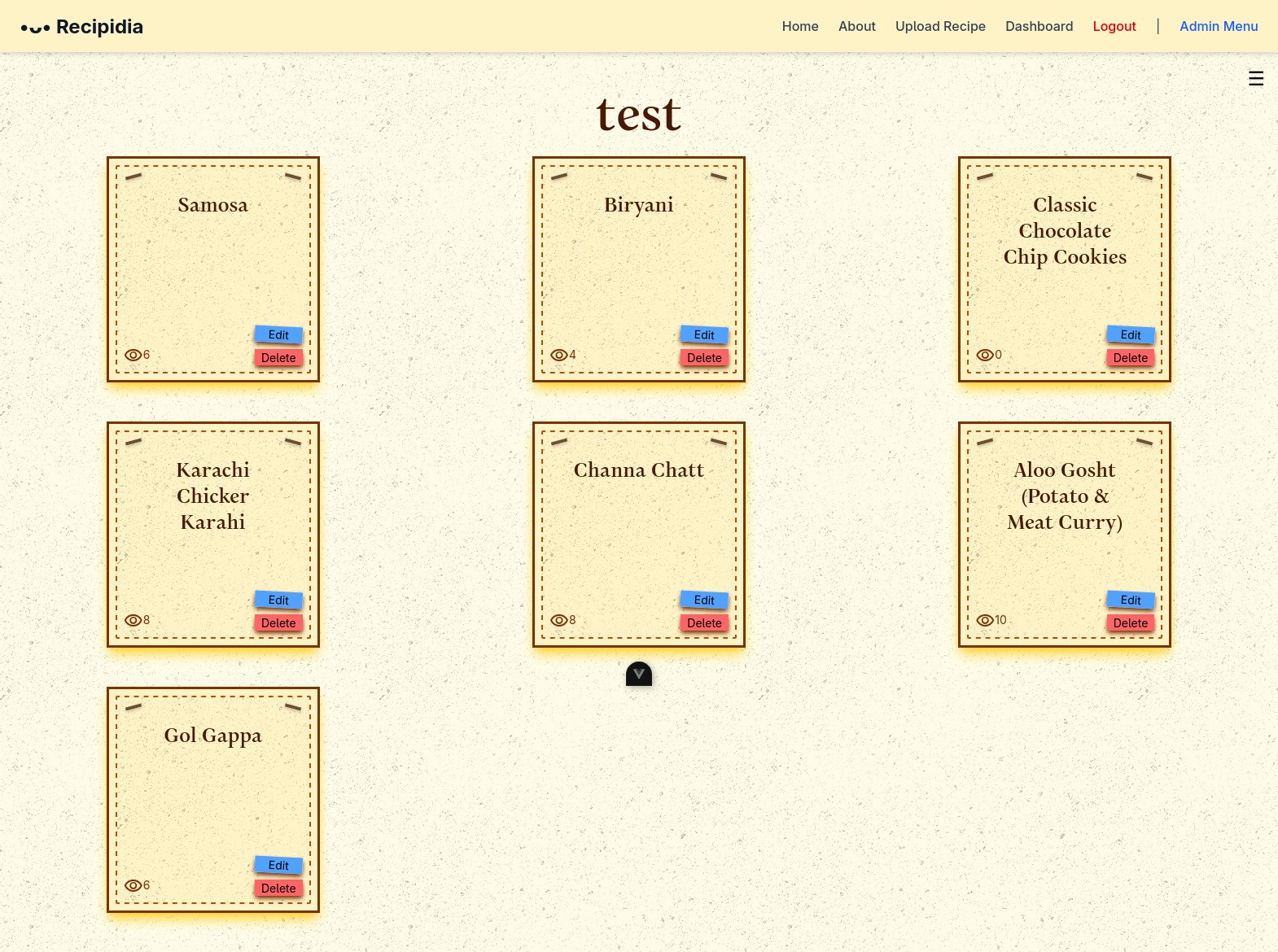
}

It integrates the itself with the user module by saving user’s data as forign key in it’s own tables.

1. Integration Testing Following is the working of Login Module
   1. Type the name and password in the feild.



* 1. Then click on the Login button.
  2. The user would be logged in.



### Admin Login

Following is the api handler for admin login

func (api \*api) VerifyAdmin(w http.ResponseWriter, r \*http.Request) { var adminReq AdminRequest

err := json.NewDecoder(r.Body).Decode(&adminReq)

ad := modals.NewAdmin()

if !(ad.CheckPassword(adminReq.Password)) {

http.Error(w, "Wrong Password", http.StatusUnauthorized) return

}

db := api.db

session := api.createAdminCookie(w) err = db.AddAdminSession(session) if err != nil {

panic(err)

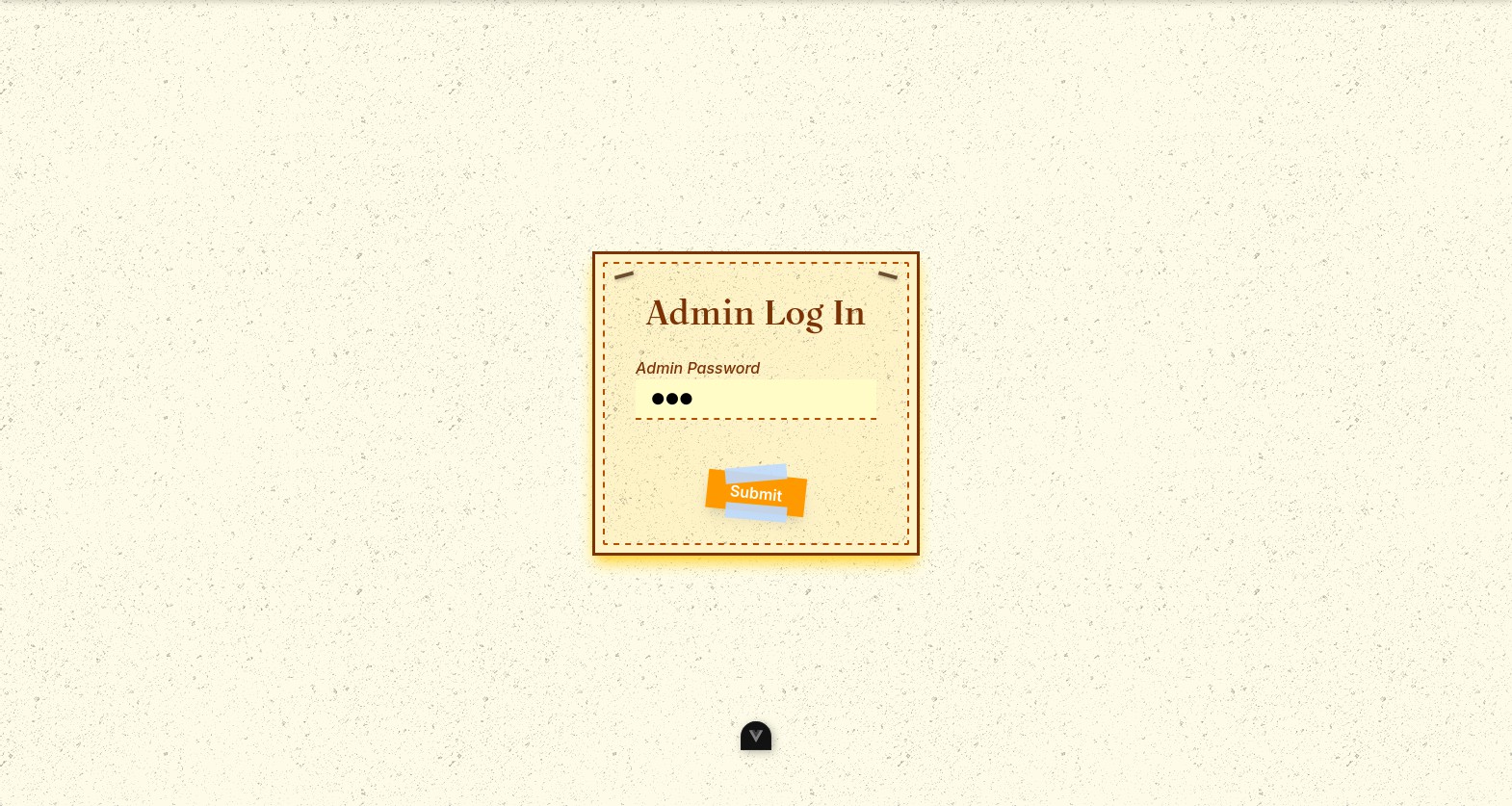
}

fmt.Printf("\nSession created and cookie set. Session ID: %s\n", session.SessionId) http.Redirect(w, r, "/view/admin-dashboard", 302)

}

It integrates the itself with the admin’s module by saving user’s data as forign key in it’s own tables.

* + - 1. Integration testing Following is the working of Login Module
         1. Type the password in the feild.



1. Then click on the Login button.
2. The admin would be logged in.

