PEEK A BOOK

ESTD 2022

Ghostwriter

ANONYMOUS BOOK CLUB

GhostWriter Portfolio

GhostWriter: Development Tool and Language

Front-end and Back-end

JavaScript

Next.js

Visual Studio

CSS3

Data Base

My SQL

MongoDB

Server and Repository

GitHub

GhostWriter: Concept

Logo

Operating under the concept of shy ghosts, GhostWriter ensures anonymity, allowing members to freely recommend books and write book diaries. The club employs a cute ghost image logo to maintain its friendly atmosphere.

Color

#FF6100

#FFFFFF

#000000

When writing book diaries, users can choose colors that match their own diaries. However, to maintain simplicity, the website primarily utilizes neutral colors, with the exception of the accent color #FF6100.

Font

ABC123

ABC123

To give a simple and classic feel, the default fonts used are Bodoni 72 and Distillery Display.

GhostWriter: Mock-up

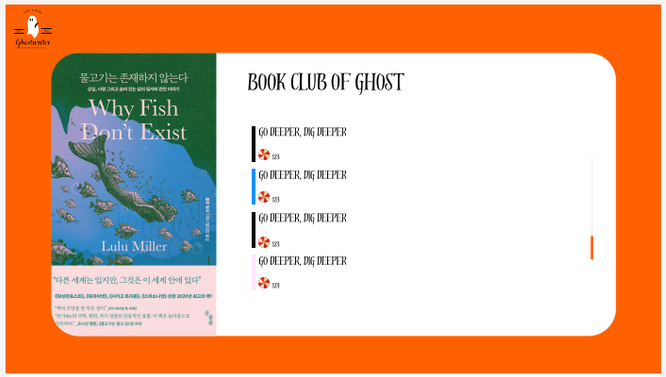
Main page

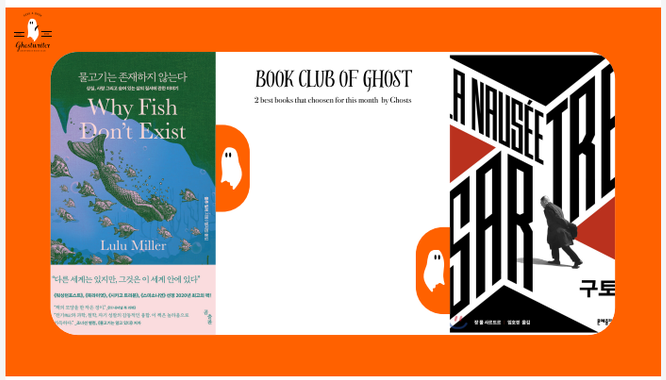
Main page and navigator

Sign In and log In

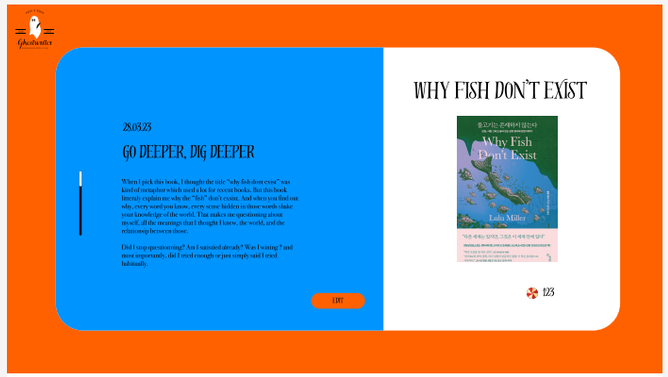
GhostWriter: Mock-up

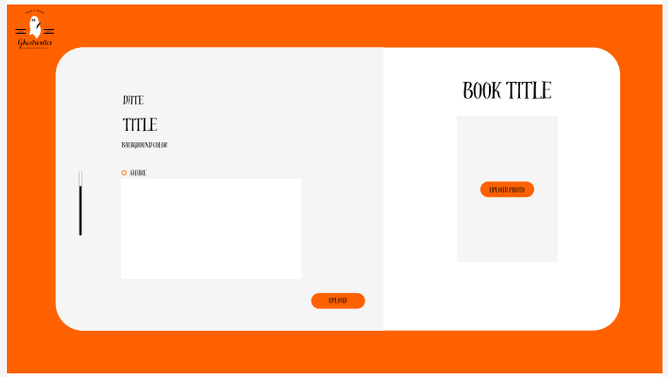
This month's book page

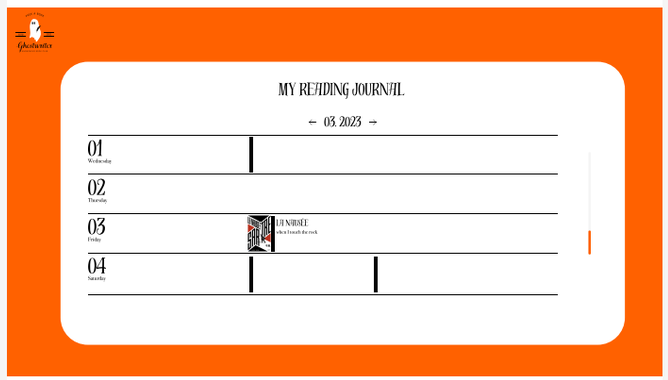




Journal page



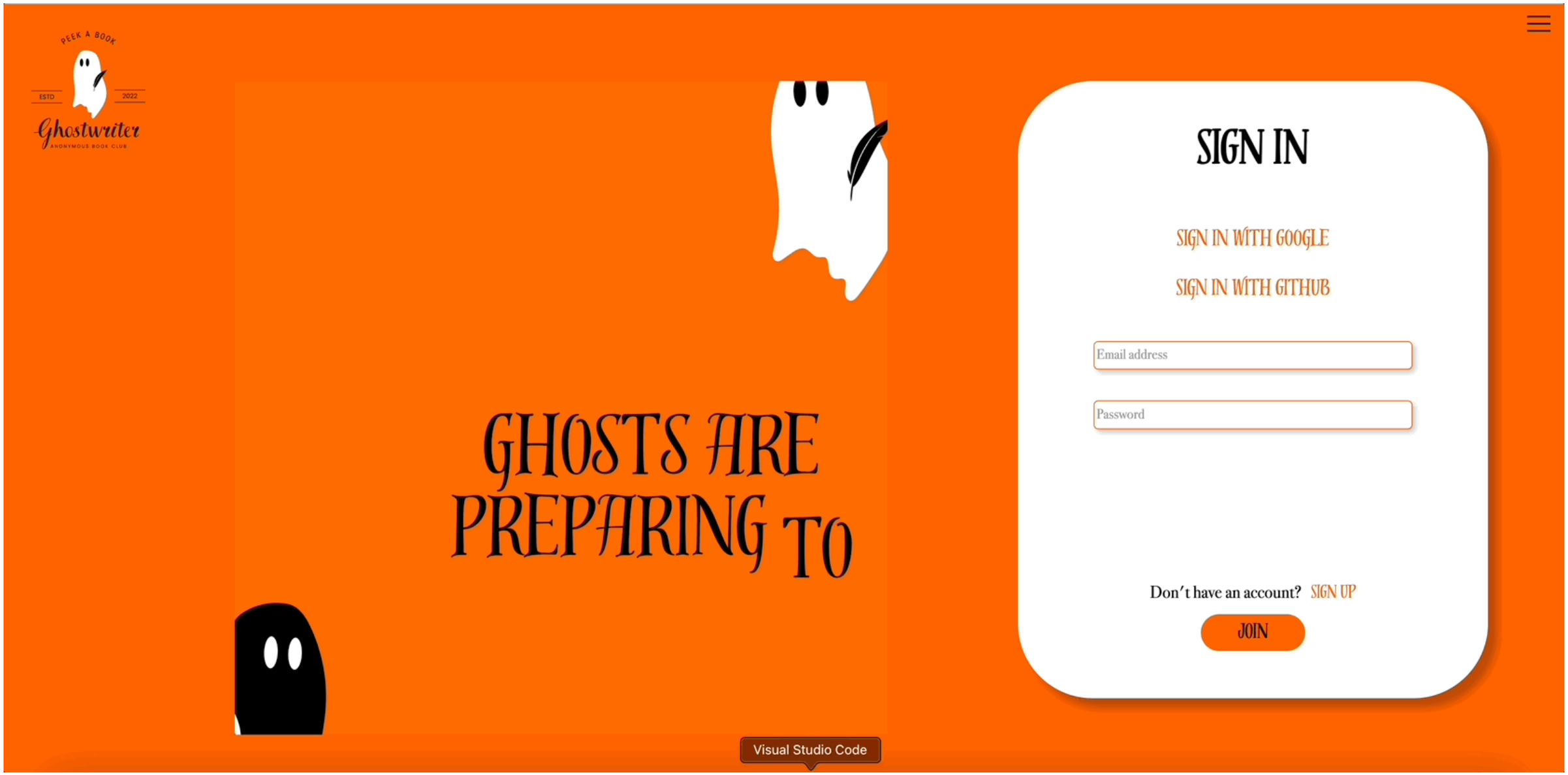




GhostWriter: Main Page

The main page focuses on showcasing the site's concept using various JavaScript effects. It includes a greeting, recommended books for the month, and instructions on how to use the site.

GhostWriter: Sign in and Log in



The site allows for both internal registration as well as integration with Google and GitHub APIs for sign-up. Keeping with the concept of anonymity, the registration process is designed to require minimal information.

GhostWriter: My Journal

The site allows for both internal registration as well as integration with Google and GitHub APIs for sign-up. Keeping with the concept of anonymity, the registration process is designed to require minimal information.

GhostWriter: Code

Main Page

```
"use client";
import React from "react";
import Image from "next/image";
import { Swiper, SwiperSlide } from "swiper/react";
import "swiper/css";
import "swiper/css/scrollbar";
import "./swiper.css";
import "./globals.css";
import { Scrollbar, Pagination, Keyboard } from "swiper/modules";
import Slide01 from "./index/Slide01";
import Slide02 from "./index/Slide02";
import Slide03 from "./index/Slide03";

export default function App() {
  return (
    <>
      <Image
        src="/images/Ghostwriter.svg"
        alt="logo_image"
        width={250}
        height={250}
        className="mainLogo"
      />
      <Swiper
        scrollbar={{ draggable: true, forceToAxis: true }}
        modules={[Scrollbar, Pagination, Keyboard]}
        pagination={true}
        keyboard={true}
        direction={"vertical"}
        className="mySwiper"
      >
        <SwiperSlide>
          <Slide01 />
        </SwiperSlide>
        <SwiperSlide>
          <Slide02 />
        </SwiperSlide>
        <SwiperSlide>
          <Slide03 />
        </SwiperSlide>
      </Swiper>
    </>
  );
}
```

To implement importing for dividing the screen into three parts with swiper module, receiving child components, and enabling drag and scroll functionality using modules and scrollbar properties.

GhostWriter: Code

Main Page: Slide01

```
"use client";

import React, { useState, useEffect } from "react";
import Image from "next/image";
import "swiper/css";
import "swiper/css/scrollbar";
import "../swiper.css";

export default function Slide01() {
  const [subtitle1Visible, setSubtitle1Visible] = useState(false);
  const [subtitle2Visible, setSubtitle2Visible] = useState(false);
  const [isMouseOver, setIsMouseOver] = useState(false);

  const handleMouseEnter = () => {
    setIsMouseOver(true);
  };
  const handleMouseLeave = () => {
    setIsMouseOver(false);
  };

  useEffect(() => {
  const timer1 = setTimeout(() => {
    setSubtitle1Visible(true);
  }, 500);

  const timer2 = setTimeout(() => {
    setSubtitle2Visible(true);
  }, 1000);

  return () => {
    clearTimeout(timer1);
    clearTimeout(timer2);
  };
}, []);
return (
  <div className="first_slide">
    <div className="title_box">
      <p className={`sub_title_1 ${subtitle1Visible ? "visible" : ""}`}>
        Welcome to book club of ghost
      </p>
      <p className={`sub_title_2 ${subtitle2Visible ? "visible" : ""}`}>
        Are you ready to peek a book?
      </p>
    </div>
    <Image
      src="/images/mainGhost.svg"
      alt="main_ghost"
      width={650}
      height={650}
      className={`first_slide_img ${isMouseOver ? "fade-out" : ""}`}
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}
    />
  </div>
);
```

Using the ‘useState’ hook to manage the states of ‘subtitle1Visible’, ‘subtitle2Visible’, and ‘isMouseOver’, which are used to track whether specific parts of the page are visible and whether the mouse is hovering over the image.

The ‘useEffect’ hook is used to sequentially change the ‘subtitle1Visible’ and ‘subtitle2Visible’ states when the component mounts to achieve a partial appearing effect.

The ‘handleMouseEnter’ function is called when the mouse moves over the image, and the ‘handleMouseLeave’ function is called when the mouse moves out of the image to implement image behavior.

Timers are used to change the ‘subtitle1Visible’ and ‘subtitle2Visible’ states at regular intervals to create a gradual disappearing and appearing effect for the image.

GhostWriter: Code

SignIn

```
"use client";
import "./signin.css";
import "../globals.css";
import Image from "next/image";
import Link from "next/link";
import AuthBtn from "./AuthBtn";
import Video from "next-video";
import React, { useRef } from "react";
import { signIn } from "next-auth/react";

export default function Signin() {
  const emailRef = useRef(null);
  const passwordRef = useRef(null);

  const handleSubmit = async () => {
  const result = await signIn("credentials", {
      email: emailRef.current,
      password: passwordRef.current,
      redirect: true,
      callbackUrl: "/",
    });
};
return (
  <div>
    <Link href="/">
      <Image
        src="/images/Ghostwriter.svg"
        alt="logo_image"
        width={250}
        height={250}
        className="logo"
      />
    </Link>
    <div className="login_box">
      <h4 className="title">SIGN IN</h4>
      <AuthBtn />
      <form method="POST" action="/api/auth/login" className="form_box_s">
        <input
          id="email"
          name="email"
          type="text"
          required
          placeholder="Email address"
          className="input_email_s"
          ref={emailRef}
          onChange={(e) => {
            emailRef.current = e.target.value;
          }}
        />
        <input
          id="password"
          name="password"
          type="password"
          required
          placeholder="Password"
          className="input_password_s"
          ref={passwordRef}
          onChange={(e) => {
            passwordRef.current = e.target.value;
          }}
        />
        <button className="btn_submit" onClick={handleSubmit}>
          JOIN
        </button>
      </form>
      <div className="signin_box">
        <span>Don't have an account?   </span>
        <Link href="/register" className="signin_link">
          SIGN UP
        </Link>
      </div>
    </div>
  );
}
```

The ‘handleSubmit’ function is called when the login form is submitted, and it authenticates the user and logs them in using the signIn function from [...nextAuth].js. The ‘signIn’ function authenticates the user using the credentials provider, attempting authentication with the provided email and password, and setting the redirect option to true to automatically redirect the user after logging in, while also specifying the callbackUrl to navigate to after redirection.

Each input field tracks its value using ref, allowing the captured input values to be used in the ‘handleSubmit’ function through onChange event handlers.

GhostWriter: Code

SignIn

```
import { connectDB } from "@/util/database";
import { MongoDBAdapter } from "@next-auth/mongodb-adapter";
import NextAuth from "next-auth";
import GithubProvider from "next-auth/providers/github";
import GoogleProvider from "next-auth/providers/google";
import CredentialsProvider from "next-auth/providers/credentials";
import bcrypt from "bcrypt";

export const authOptions = {
  providers: [
    GithubProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    }),

    GoogleProvider({
      clientId: process.env.GITHUB_CLIENT_SECRET,
      clientSecret: process.env.GITHUB_CLIENT_SECRET,
    }),

  CredentialsProvider({
    // credentials provider 설정
    name: "Credentials",
    credentials: {
      email: { label: "email", type: "text" },
      password: { label: "password", type: "password" },
    },
    async authorize(credentials) {
      let db = (await connectDB).db("ghostwriter");
      let user = await db
        .collection("member")
        .findOne({ email: credentials.email });
      if (!user) {
        console.log("해당 이메일은 없음");
        return null;
      }
      const pwcheck = await bcrypt.compare(
        credentials.password,
        user.password
      );
      if (!pwcheck) {
        console.log("비번틀림");
        return null;
      }
      return user;
    },
  }),
],

//3. jwt 써놔야 잘됩니다 + jwt 만료일설정
session: {
  strategy: "jwt",
  maxAge: 30 * 24 * 60 * 60, //30일
},
```

```
session: {
    strategy: "jwt",
    maxAge: 30 * 24 * 60 * 60, //30일
  },

  callbacks: {
    //4. jwt 만들 때 실행되는 코드
    //user변수는 DB의 유저정보담겨있고 token.user에 뭐 저장하면 jwt에 들어갑니다.
    jwt: async ({ token, user }) => {
      if (user) {
        token.user = {};
        token.user.name = user.name;
        token.user.email = user.email;
        token.user.role = user.role;
      }
      return token;
    },
    //5. 유저 세션이 조회될 때 마다 실행되는 코드
    session: async ({ session, token }) => {
      session.user = token.user;
      return session;
    },
  },
  pages: {
    signIn: "/signin",
    logIn: "/login",
  },
  secret: process.env.NEXTAUTH_SECRET,
  adapter: MongoDBAdapter(connectDB),
};
export default NextAuth(authOptions);
```

This code sets up options for configuring authentication using NextAuth. It configures the 'providers' to specify which authentication providers to use, including GitHub, Google, and the Credentials provider. The 'CredentialsProvider' is configured to authenticate users using their email and password. In the 'authorize' function of this provider, it searches for the user in the database using the provided email and password, performs authentication, and returns the user object once authenticated.

GhostWriter: Code

SignIn

```
import { connectDB } from "@/util/database";
import bcrypt from "bcrypt";

export default async function handler(req, res) {
  const db = (await connectDB).db("ghostwriter");
  let email = await db.collection("member").findOne({ email:
req.body.email });
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  if (req.method === "POST") {
    if (
      req.body.name === "" ||
      req.body.email === "" ||
      req.body.password === ""
    ) {
      res.write(
        "<script>alert('Hostname, Email address, and password are
required')</script>"
      );
      res.write("<script>window.location='../../../register'</script>");
      return res.end(); // 응답 종료
    } else {
      if (!emailRegex.test(req.body.email)) {
        res.write("<script>alert('Invalid email address format')</script>");
        res.write("<script>window.location='../../../register'</script>");
        return res.end(); // 응답 종료
      } else if (email != null) {
        res.write(
          "<script>alert('Your email address is already taken')</script>"
        );
        res.write("<script>window.location='../../../register'</script>");
        return res.end(); // 응답 종료
      } else if (req.body.password !== req.body.confirm_password) {
        res.write(
          "<script>alert('Please check your confirm password')</script>"
        );
        res.write("<script>window.location='../../../register'</script>");
        return res.end(); // 응답 종료
      } else {
        delete req.body.confirm_password;
        let hash = await bcrypt.hash(req.body.password, 10);
        req.body.password = hash;
        let db = (await connectDB).db("ghostwriter");
        await db.collection("member").insertOne(req.body);
        return res.redirect(302, "/"); // 바로 리다이렉트
      }
    }
  }
}
```

The code serves as a Next.js API route handler, performing the functionality of securely processing and validating user-submitted registration form data to facilitate safe registration. It encompasses the steps of email verification, email format validation, password confirmation, password hashing, and subsequent redirection upon successful registration.

It verifies if the email submitted through the registration form already exists in the database. If it does, indicating that the email is already in use by another user, registration is denied. It checks if the submitted email adheres to the correct format, validating it using regular expressions. It verifies whether the submitted password matches the password confirmation and enhances security by hashing the password. Using the bcrypt library, it hashes the password and stores the hashed password in the database. Upon successfully passing all validation steps, it stores the submitted information in the database to complete the registration process and redirects the user to the homepage.

GhostWriter: Code

Register

```
import { connectDB } from "@/util/database";
import bcrypt from "bcrypt";
import { signIn } from "next-auth/react";

export default async function handler(req, res) {
  const db = (await connectDB).db("ghostwriter");

  if (req.method === "POST") {
    const { email, password } = req.body;

    if (email === "" || password === "") {
      res.write(
        "<script>alert('Please enter your email address and password')</script>"
      );
      res.write("<script>window.location='../../../signin'</script>");
      return res.end();
    }

    // 사용자 이메일로 검색
    const user = await db.collection("member").findOne({ email });
    if (!user) {
      res.write("<script>alert('Invalid email address')</script>");
      res.write("<script>window.location='../../../signin'</script>");
      return res.end();
    }

    // 비밀번호 확인
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      res.write("<script>alert('Invalid password')</script>");
      res.write("<script>window.location='../../../signin'</script>");
      return res.end();
    }

    // 인증이 성공하면 메인 페이지로 이동
    console.log(user);
    res.redirect(302, "/");
    return res.end();
  } else {
    // POST 요청이 아닌 경우 405 Method Not Allowed 응답
    res.writeHead(405);
    return res.end();
  }
}
```

This code verifies and securely handles login form data submitted by users.

It connects to the database and checks if the request method is POST. If it's a POST request, it extracts the email and password from the request. If either the email or password is empty, it displays a warning prompting the user to enter their email address and password and redirects to the login page.

It searches for the user in the database using the email. If the user is not found, it notifies that the email address is invalid and redirects to the login page. If the user is found, it compares the entered password with the hashed password in the database to verify if they match. If the passwords don't match, it notifies that the password is invalid and redirects to the login page.

If authentication is successful, it redirects the user to the main page; otherwise, it returns a 405 Method Not Allowed response.

GhostWriter: Code

MyJournal

```
"use client";

import "../globals.css";
import Image from "next/image";
import Link from "next/link";
import React, { useState } from "react";
import "react-calendar/dist/Calendar.css";
import "./myJournal.css";

import {
  format, subMonths, addMonths, startOfWeek, addDays, isSameDay, lastDayOfWeek, getWeek,
addWeeks, subWeeks,} from "date-fns";

const Calendar = ({ showDetailsHandle }) => {
  const [currentMonth, setCurrentMonth] = useState(new Date());
  const [currentWeek, setCurrentWeek] = useState(getWeek(currentMonth));
  const [selectedDate, setSelectedDate] = useState(new Date());

  const changeMonthHandle = (btnType) => {
    if (btnType === "prev") {
      setCurrentMonth(subMonths(currentMonth, 1));
    }
    if (btnType === "next") {
      setCurrentMonth(addMonths(currentMonth, 1));
    }
  };

  const changeWeekHandle = (btnType) => {
    if (btnType === "prev") {
      setCurrentMonth(subWeeks(currentMonth, 1));
      setCurrentWeek(getWeek(subWeeks(currentMonth, 1)));
    }
    if (btnType === "next") {
      setCurrentMonth(addWeeks(currentMonth, 1));
      setCurrentWeek(getWeek(addWeeks(currentMonth, 1)));
    }
  };

  const onDateClickHandle = (day, dayStr) => {
    setSelectedDate(day);
    showDetailsHandle(dayStr);
  };

  const renderHeader = () => {
    const dateFormat = "MMM yyyy";
    return (
      <div className="header row flex-middle">
        <div className="col col-start">
        </div>
        <div className="col col-center">
          <span>{format(currentMonth, dateFormat)}</span>
        </div>
        <div className="col col-end">
        </div>
      </div>
    );
  };
```

```
  const renderCells = () => {
    const startDate = startOfWeek(currentMonth, { weekStartsOn: 1 });
    const endDate = lastDayOfWeek(currentMonth, { weekStartsOn: 1 });
    const dateFormat = "d";
    const dateFormat2 = "EEEE";
    const rows = [];
    let days = [];
    let day = startDate;
    let formattedDate = "";

    while (day <= endDate) {
      for (let i = 0; i < 7; i++) {
        formattedDate = format(day, dateFormat);
        const cloneDay = day;
        days.push(
          <div
            className={`col cell ${
              isSameDay(day, new Date())
                ? "today"
                : isSameDay(day, selectedDate)
                ? "selected"
                : ""
            }`}
            key={day}
            onClick={() => {
              const dayStr = format(cloneDay, "ccc dd MMM yy");
              onDateClickHandle(cloneDay, dayStr);
            }}
          >
            <div className="date">
              <span className="number">{formattedDate}</span>
              <span className="days">
                {format(addDays(startDate, i), dateFormat2)}
              </span>
            </div>
          </div>
        );
        day = addDays(day, 1);
      }

      rows.push(
        <div className="row" key={day}>
          {days}
        </div>
      );
      days = [];
    }
    return <div className="body">{rows}</div>;
  };
  const renderWeek = () => {
    return (
      <div className="header rowBottom flex-middle">
        <div className="col col-start">
          <div className="icon" onClick={() => changeWeekHandle("prev")}>
            prev week
          </div>
        </div>
        <div className="col col-end" onClick={() => changeWeekHandle("next")}>
          <div className="icon">next week</div>
        </div>
      </div>
    );
  };
```

GhostWriter: Code

MyJournal

```
  return (
    <div>
      <Link href="/">
        <Image
          src="/images/Ghostwriter.svg"
          alt="logo_image"
          width={250}
          height={250}
          className="logo"
        />
      </Link>
      <div className="calendar-box">
        <h4 className="title">My Reading myJournal</h4>
        <div className="calendar">
          {renderWeek()}
          {renderHeader()}
          {renderCells()}
        </div>
      </div>
    </div>
  );
};
```

This code represents a calendar component (Calendar) implemented in React.

The component manages state using the useState hook to track the current month (currentMonth), current week (currentWeek), and selected date (selectedDate).

It provides functions changeMonthHandle and changeWeekHandle to navigate between months and weeks, respectively, based on the user's interaction.

The onDateClickHandle function is triggered when a user clicks on a date cell in the calendar. It updates the selected date and invokes a callback function (showDetailsHandle) with the selected date string as an argument.

The renderHeader function renders the header section of the calendar, displaying the current month and providing navigation options to switch between months.

The renderCells function generates the grid cells for displaying dates in the calendar. It creates rows and columns of date cells, with each cell representing a day in the current month. It also highlights the current day and the selected day.

The renderWeek function renders a control for navigating between weeks, allowing users to view the calendar by week.

Finally, the Calendar component is exported as the default export, making it available for use in other components. It also includes JSX for rendering the logo, title, and the calendar itself within a parent container.