Indian Institute of Technology, Kharagpur
Department of Computer Science and Engineering

Design Lab (CS59001) Documentation

Sarvagya: Real-time Education Platform for Seamless Student-Instructor Collaboration

10th April, 2025

Hardik Soni
20CS30023

Contents

1. Abstract

Sarvagya is an innovative educational platform designed to bridge the gap between students and instructors through seamless digital interaction. The platform leverages modern web technologies to create an immersive learning environment where knowledge seekers can connect with subject matter experts in real-time. Sarvagya facilitates communication through integrated chat and video conferencing, personalized content delivery through targeted announcements, and community building through a follow-based network system. Built on Next.js, TypeScript, and Supabase, with real-time capabilities powered by Stream's SDK, Sarvagya represents a comprehensive solution to the challenges of remote education and mentorship in the digital age. The platform's intuitive interface and robust feature set create a dynamic ecosystem where educational relationships can flourish beyond the constraints of physical classrooms.

2. Introduction

2.1 Background and Motivation

The digital transformation of education has expanded access to learning resources but has often fallen short in facilitating meaningful connections between students and instructors. Sarvagya addresses this gap by creating a specialized platform where educational relationships can flourish beyond physical limitations. Motivated by the need to democratize access to expertise, enhance personalized learning experiences, and overcome the engagement challenges of remote education, Sarvagya leverages modern web technologies to build a sustainable educational ecosystem. The platform responds to the lessons learned during the global shift to remote learning, which highlighted the inadequacy of general-purpose communication tools for educational contexts. By integrating targeted features for real-time interaction, content personalization, and community building, Sarvagya aims to transform how knowledge is shared and relationships are formed in digital educational environments, making quality instruction and mentorship accessible regardless of geographic or socioeconomic barriers.

3. Problem Statement

3.1 Overview

Despite the proliferation of digital learning platforms, there exists a significant gap in facilitating meaningful educational relationships between students and instructors in virtual environments. Current solutions often prioritize content delivery over interactive engagement, resulting in impersonal learning experiences that fail to replicate the benefits of direct mentorship. Sarvagya addresses this challenge by developing an integrated platform that enables personalized connections, real-time communication, and targeted knowledge sharing between students and instructors, thereby creating a more effective and engaging digital educational ecosystem.

3.2 Key Challenges to Address

- **Limited Personalization:** Existing platforms offer minimal customization of learning experiences based on individual student interests and needs.
- **Communication Barriers:** Current solutions lack seamless integration of multiple communication modalities (text, video) necessary for effective educational interaction.
- **Instructor Discovery:** Students struggle to identify and connect with instructors whose expertise aligns with their specific learning objectives and interests.
- **Community Building:** Digital learning environments often fail to foster the sense of community and relationship-building that enhances educational outcomes.
- **Content Relevance:** Broadcast-style content distribution results in information overload rather than targeted, relevant educational announcements.

- **Engagement Sustainability:** Maintaining consistent engagement between students and instructors remains challenging in virtual learning contexts.
- **Technical Fragmentation:** Educational interactions typically require multiple disconnected platforms for different aspects of the learning relationship.
- **Accessibility Barriers:** Geographic and socioeconomic factors continue to limit access to quality educational mentorship and guidance.

4. System Architecture

4.1 Overview

The Sarvagya platform architecture consists of four main components:

1. **Frontend Application:** A Next.js-based web application providing the user interface for both students and instructors, with server-side rendering for optimal performance and SEO.
2. **Authentication and Database:** Supabase provides user authentication, data storage, and real-time database capabilities to manage user profiles, relationships, and content.
3. **Real-time Communication:** Stream SDK integration enables real-time chat and video conferencing capabilities between students and instructors.
4. **Content Delivery:** A personalized content delivery system for announcements, educational materials, and notifications based on user relationships and preferences.

4.2 Component Interaction and Data Flow

The architecture components interact as follows:

- *User Authentication → Supabase Auth → User Profile Creation*
- *Student/Instructor Profiles → Supabase Database → Relationship Management*
- *Follow Requests → Database Updates → Personalized Feed Generation*
- *Announcements → Database Storage → Targeted Distribution*
- *Chat Initiation → Stream Chat SDK → Real-time Messaging*
- *Video Call Scheduling → Stream Video SDK → Interactive Sessions*

4.3 Technical Components

1. **Frontend Layer:**
   - Next.js framework with TypeScript for type safety
   - React components organized by functionality (common, instructor, student)
   - Tailwind CSS with ShadCn UI for responsive design
   - Client-side state management for real-time interactions
2. **Backend Services:**
   - Next.js API routes for server-side operations
   - Server Actions for data mutations and business logic
   - Supabase Row Level Security for data access control
   - Middleware for authentication and request processing
3. **Data Storage:**

- Supabase PostgreSQL database for structured data
- Supabase Storage for file management (profile images, educational materials)
- Stream Chat storage for message history
- Stream Video for call recordings and session data

4. **Integration Layer:**
   - Supabase client libraries for database operations
   - Stream SDK for real-time communication
   - RESTful API patterns for external integrations
   - Webhook support for event-driven architecture

4.4 Data Models

The core data models in the system include:

- *User* → Base entity with authentication details
- *Student* → Extended user with following list and interests
- *Instructor* → Extended user with professional details and followers
- *Announcement* → Content shared by instructors to followers
- *Channel* → Communication space between users
- *Call* → Scheduled or ad-hoc video sessions

4.5 Requirements

4.5.1 Sarvagya Build System Setup

For a brief video tutorial refer here.

4.6 Data Flow Diagram

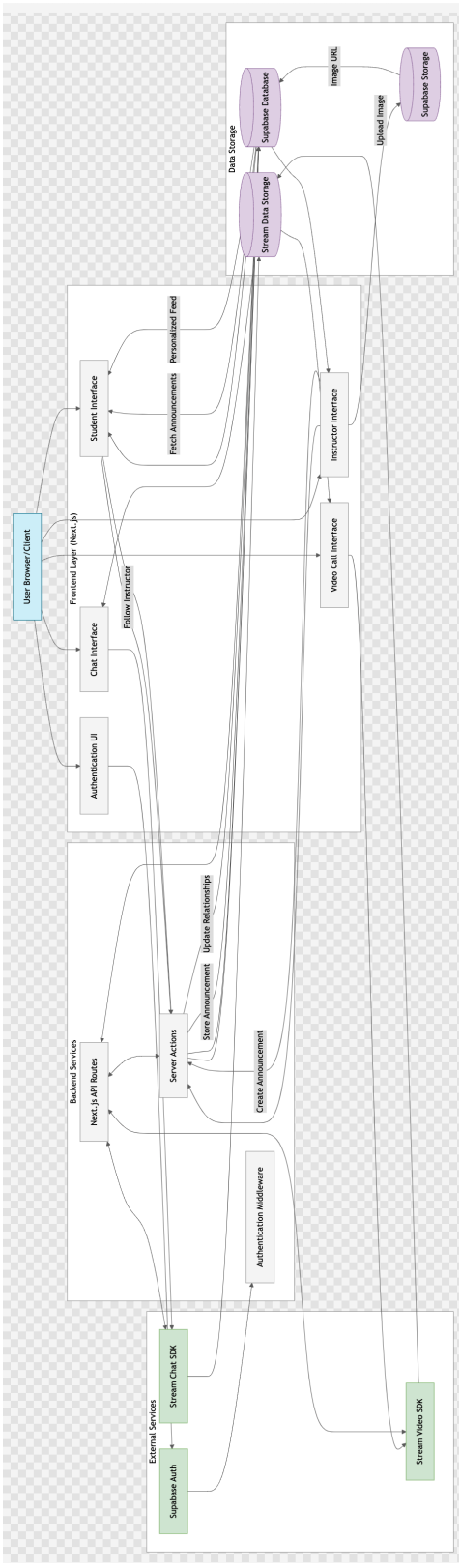

Figure 1: Data Flow Diagram of system showing broad components

5. Implementation Details

5.1 Technology Stack

The Sarvagya platform is built using a modern technology stack that enables real-time communication, secure authentication, and responsive user interfaces:

- **Frontend**: Next.js 15 with TypeScript and Tailwind CSS
- **Backend**: Next.js API routes and Server Actions
- **Authentication**: Supabase Authentication
- **Database**: Supabase PostgreSQL
- **Storage**: Supabase Storage
- **Real-time Communication**: Stream Chat and Video SDKs
- **UI Components**: ShadCn UI
- **Testing**: Jest and React Testing Library

5.2 Core Components

The implementation is structured around several core components that work together to provide a seamless experience for both students and instructors.

5.2.1 Authentication System

The authentication system leverages Supabase's secure authentication services to handle user registration, login, and session management for both students and instructors.

```
"use server";
import { createClient } from "@/lib/supabase/server";

export async function studentSignUp(formData: FormData) {
  const supabase = await createClient();
  const credentials = {
    email: formData.get("email") as string,
    password: formData.get("password") as string,
    interest: formData.get("interest") as string,
    name: formData.get("name") as string,
  };

  const { data, error } = await supabase.auth.signUp({
    email: credentials.email,
    password: credentials.password,
    options: {
      data: {
        interest: credentials.interest,
        name: credentials.name,
      },
    },
  });

  if (error) {
    return { error: error.message, status: error.status, user: null };
  } else if (data.user?.identities?.length === 0) {
    return { error: "User already exists", status: 409, user: null };
  }

  return { error: null, status: 200, user: data.user };
```

```
}
```

Listing 1: Authentication Server Action

5.2.2 User Profile Management

The platform maintains separate profiles for students and instructors, with different fields and capabilities for each role.

```
export interface InstructorProps {
  name: string;
  bio: string;
  email: string;
  image: string;
  occupation: string;
  url: string;
  id: string;
  followers: string[];
}

export interface ProfileProps {
  name: string;
  bio: string;
  email: string;
  image: string;
  occupation: string;
  url: string;
}
```

Listing 2: User Profile Types

5.2.3 Real-time Chat System

The chat functionality is implemented using Stream's Chat SDK, allowing students and instructors to communicate in real-time.

```
export async function createChannel({
  userId,
  data,
}: {
  userId: string;
  data: { name: string; imageUrl: string };
}) {
  try {
    const channels = await serverClient.queryChannels(
      {
        members: { $in: [userId] },
        type: "messaging",
      },
      { last_message_at: -1 }
    );

    if (channels.length > 0) {
      return {
        success: false,
        error: "You already have an existing channel",
        id: channels[0].id,
      };
    }

    const channel = serverClient.channel("messaging", `channel-${userId}`, {
      name: data.name,
      image: data.imageUrl,
      members: [userId],
      created_by_id: userId,
    });
```

```
    await channel.create();
    return { success: true, error: null, id: channel.id };
  } catch (err) {
    return { success: false, error: "Failed to create channel", id: null };
  }
}
```

Listing 3: Chat Channel Creation

5.2.4 Video Call Integration

The platform integrates Stream's Video SDK to enable face-to-face interactions between students and instructors.

```
import { useEffect, useState } from "react";
import { Call, useStreamVideoClient } from "@stream-io/video-react-sdk";

export const useGetCallById = (id: string | string[]) => {
  const [call, setCall] = useState<Call>();
  const [isCallLoading, setIsCallLoading] = useState(true);

  const client = useStreamVideoClient();

  useEffect(() => {
    if (!client) return;

    const loadCall = async () => {
      try {
        const { calls } = await client.queryCalls({
          filter_conditions: { id },
        });

        if (calls.length > 0) setCall(calls[0]);

        setIsCallLoading(false);
      } catch (error) {
        console.error(error);
        setIsCallLoading(false);
      }
    };

    loadCall();
  }, [client, id]);

  return { call, isCallLoading };
};
```

Listing 4: Video Call Hook

5.2.5 Content Feed System

The feed system delivers personalized content to students based on the instructors they follow.

```
export const getStudentFeed = async (studentId: string) => {
  const supabase = createClient();
  const { data, error } = await supabase
    .from("students")
    .select("following_list")
    .eq("id", studentId);

  if (error || !data) {
    return null;
  }
  const followingList = data[0].following_list;
  if (followingList.length === 0) {
    return [];
  }
```

```
  const { data: announcementsData, error: announcementsError } = await supabase
    .from("announcements")
    .select("*")
    .in("author_id", followingList)
    .order("created_at", { ascending: false });

  if (announcementsError || !announcementsData) {
    return null;
  }
  return announcementsData;
};
```

Listing 5: Student Feed Retrieval

5.3 Database Schema

The database schema is designed to support the relationships between students, instructors, and content.

5.3.1 Core Tables

- **students**: Stores student profiles and their following relationships
- **instructors**: Stores instructor profiles and their follower counts
- **announcements**: Stores content created by instructors

```
CREATE TABLE announcements (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  author_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,
  author_name TEXT NOT NULL,
  author_image TEXT NOT NULL,
  author_title TEXT NOT NULL,
  content TEXT NOT NULL,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  interest TEXT NOT NULL,
  likes TEXT[] DEFAULT '{}'::TEXT[]
);

Security
ALTER TABLE announcements ENABLE ROW LEVEL SECURITY;

-- Create policies
CREATE POLICY "Anyone can read announcements"
  ON announcements FOR SELECT
  USING (true);

CREATE POLICY "Authors can insert their own announcements"
  ON announcements FOR INSERT
  WITH CHECK (auth.uid() = author_id);

CREATE POLICY "Authors can update their own announcements"
  ON announcements FOR UPDATE
  USING (auth.uid() = author_id);
```

Listing 6: Database Migration for Announcements

5.4 Frontend Components

The user interface is built with reusable components that provide a consistent experience across the platform.

5.4.1 UI Component Library

The platform uses ShadCn UI components for a consistent design language.

```
import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const buttonVariants = cva(
  "inline-flex items-center justify-center gap-2 whitespace-nowrap rounded-md text-sm font-medium transition-colors focus-vis
  {
    variants: {
      variant: {
        default:
          "bg-primary text-primary-foreground shadow hover:bg-primary/90",
        destructive:
          "bg-destructive text-destructive-foreground shadow-sm hover:bg-destructive/90",
        outline:
          "border border-input bg-background shadow-sm hover:bg-accent hover:text-accent-foreground",
        secondary:
          "bg-secondary text-secondary-foreground shadow-sm hover:bg-secondary/80",
        ghost: "hover:bg-accent hover:text-accent-foreground",
        link: "text-primary underline-offset-4 hover:underline",
      },
      size: {
        default: "h-9 px-4 py-2",
        sm: "h-8 rounded-md px-3 text-xs",
        lg: "h-10 rounded-md px-8",
        icon: "h-9 w-9",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)

export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement>,
    VariantProps<typeof buttonVariants> {
  asChild?: boolean
}

const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  ({ className, variant, size, asChild = false, ...props }, ref) => {
    const Comp = asChild ? Slot : "button"
    return (
      <Comp
        className={cn(buttonVariants({ variant, size, className }))}
        ref={ref}
        {...props}
      />
    )
  }
)
Button.displayName = "Button"

export { Button, buttonVariants }
```

Listing 7: Button Component

5.4.2 Page Components

Each page in the application is composed of multiple components that work together to provide a cohesive experience.

```
import { Suspense } from "react";
import { notFound } from "next/navigation";
import { createClient } from "@/lib/supabase/server";
import AnnouncementBox from "./(components)/AnnouncementBox";
import MeetingsBox from "./(components)/MeetingsBox";

export default async function InstructorPage({
  params,
}: {
  params: { id: string };
}) {
  const supabase = await createClient();

  // Fetch instructor data
  const { data: instructor, error } = await supabase
    .from("instructors")
    .select("*")
    .eq("id", params.id)
    .single();

  if (error || !instructor) {
    notFound();
  }

  // Fetch announcements
  const { data: announcements } = await supabase
    .from("announcements")
    .select("*")
    .eq("author_id", params.id)
    .order("created_at", { ascending: false });

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
        <div className="md:col-span-1">
          <div className="bg-white rounded-lg shadow p-6">
            <div className="flex flex-col items-center">
              <img
                src={instructor.image}
                alt={instructor.name}
                className="w-32 h-32 rounded-full object-cover"
              />
              <h1 className="text-2xl font-bold mt-4">{instructor.name}</h1>
              <p className="text-gray-600">{instructor.occupation}</p>
              <p className="mt-4 text-center">{instructor.bio}</p>
              <div className="mt-4 flex items-center">
                <span className="text-gray-600 mr-2">
                  {instructor.followers.length} followers
                </span>
              </div>
            </div>
          </div>
        </div>

        <div className="md:col-span-2">
          <Suspense fallback={<div>Loading announcements...</div>}>
            <AnnouncementBox announcements={announcements || []} />
          </Suspense>

          <Suspense fallback={<div>Loading meetings...</div>}>
            <MeetingsBox instructorId={params.id} />
          </Suspense>
        </div>
      </div>
    </div>
  );
}
```

Listing 8: Instructor Profile Page

5.5 Testing Infrastructure

The project includes a comprehensive testing infrastructure to ensure code quality and reliability.

5.5.1 Test Configuration

Jest is configured to work with Next.js and the project's component structure.

```
const nextJest = require('next/jest');

const createJestConfig = nextJest({
  dir: './',
});

const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  testEnvironment: 'jest-environment-jsdom',
  moduleNameMapper: {
    // Handle module aliases
    '^@/components/(.*)$': '<rootDir>/src/components/$1',
    '^@/lib/(.*)$': '<rootDir>/src/lib/$1',
    '^@/hooks/(.*)$': '<rootDir>/src/hooks/$1',
    '^@/actions/(.*)$': '<rootDir>/src/actions/$1',
    '^@/types/(.*)$': '<rootDir>/src/types/$1',
    '^@/constants/(.*)$': '<rootDir>/src/constants/$1',
    '^@/config/(.*)$': '<rootDir>/src/config/$1',
  },
  testPathIgnorePatterns: ['<rootDir>/node_modules/', '<rootDir>/.next/'],
  transform: {
    // Use babel-jest to transpile tests with the next/babel preset
    '^.+\\.(js|jsx|ts|tsx)$': ['babel-jest', { presets: ['next/babel'] }],
  },
  transformIgnorePatterns: [
    '/node_modules/',
    '^.+\\.module\\.(css|sass|scss)$',
  ],
  collectCoverage: true,
  collectCoverageFrom: [
    'src/**/*.{js,jsx,ts,tsx}',
    '!src/**/*.d.ts',
    '!src/**/*.stories.{js,jsx,ts,tsx}',
    '!src/pages/_app.tsx',
    '!src/pages/_document.tsx',
    '!**/node_modules/**',
  ],
};

module.exports = createJestConfig(customJestConfig);
```

Listing 9: Jest Configuration

5.5.2 Component Tests

Components are tested to ensure they render correctly and handle user interactions as expected.

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import { Button } from '../button';

describe('Button Component', () => {
  it('renders correctly with default props', () => {
    render(<Button>Click me</Button>);

    const button = screen.getByRole('button', { name: /click me/i });
    expect(button).toBeInTheDocument();
    expect(button).toHaveClass('bg-primary');
  });
```

```
  it('renders with different variants', () => {
    const { rerender } = render(<Button variant="destructive">Destructive</Button>);

    let button = screen.getByRole('button', { name: /destructive/i });
    expect(button).toHaveClass('bg-destructive');

    rerender(<Button variant="outline">Outline</Button>);
    button = screen.getByRole('button', { name: /outline/i });
    expect(button).toHaveClass('border-input');

    rerender(<Button variant="secondary">Secondary</Button>);
    button = screen.getByRole('button', { name: /secondary/i });
    expect(button).toHaveClass('bg-secondary');
  });

  it('applies additional className when provided', () => {
    render(<Button className="custom-class">Custom Class</Button>);

    const button = screen.getByRole('button', { name: /custom class/i });
    expect(button).toHaveClass('custom-class');
    expect(button).toHaveClass('bg-primary'); // Still has the default classes
  });
});
```

Listing 10: Button Component Test

5.6 Deployment Configuration

The application is configured for deployment to various environments with appropriate environment variables.

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    domains: [
      'api.dicebear.com',
      'localhost',
      'your-supabase-project.supabase.co',
    ],
  },
  experimental: {
    serverActions: true,
  },
  env: {
    NEXT_PUBLIC_SUPABASE_URL: process.env.NEXT_PUBLIC_SUPABASE_URL,
    NEXT_PUBLIC_SUPABASE_ANON_KEY: process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY,
    STORAGE_URL: process.env.STORAGE_URL,
    NEXT_PUBLIC_STREAM_API_KEY: process.env.NEXT_PUBLIC_STREAM_API_KEY,
    STREAM_SECRET_KEY: process.env.STREAM_SECRET_KEY,
    NEXT_PUBLIC_PAGE_URL: process.env.NEXT_PUBLIC_PAGE_URL,
    NEXT_PUBLIC_STREAM_CHANNEL_IMAGE_URL: process.env.NEXT_PUBLIC_STREAM_CHANNEL_IMAGE_URL,
  },
};

module.exports = nextConfig;
```

Listing 11: Next.js Configuration

5.7 Security Considerations

The implementation includes several security measures to protect user data and prevent unauthorized access.

5.7.1 Row Level Security

Supabase's Row Level Security is used to control access to database records based on user roles and ownership.

```
-- Enable RLS on students table
ALTER TABLE students ENABLE ROW LEVEL SECURITY;

-- Students can read all student profiles
CREATE POLICY "Students can view all profiles"
  ON students FOR SELECT
  USING (true);

-- Students can only update their own profile
CREATE POLICY "Students can update own profile"
  ON students FOR UPDATE
  USING (auth.uid() = id);

-- Enable RLS on instructors table
ALTER TABLE instructors ENABLE ROW LEVEL SECURITY;

-- Anyone can view instructor profiles
CREATE POLICY "Anyone can view instructor profiles"
  ON instructors FOR SELECT
  USING (true);
```

Listing 12: Row Level Security Policies

true);

– Instructors can only update their own profile CREATE POLICY "Instructors can update own profile" ON instructors FOR UPDATE USING (auth.uid() = id);

5.7.2 Authentication Middleware

The application uses middleware to protect routes and redirect unauthenticated users.

```
import { NextResponse, type NextRequest } from 'next/server'
import { createServerClient } from '@supabase/ssr'

export async function updateSession(request: NextRequest) {
  let supabaseResponse = NextResponse.next({
    request,
  })

  const supabase = createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {
      cookies: {
        getAll() {
          return request.cookies.getAll()
        },
        setAll(cookiesToSet) {
          cookiesToSet.forEach(({ name, value }) => request.cookies.set(name, value))
          supabaseResponse = NextResponse.next({
            request,
          })
          cookiesToSet.forEach(({ name, value, options }) =>
            supabaseResponse.cookies.set(name, value, options)
          )
        },
      },
    }
  )

  const {
    data: { user },
  } = await supabase.auth.getUser()
```

```
  if (
    !user &&
    request.nextUrl.pathname !== "/" &&
    !request.nextUrl.pathname.startsWith('/instructor/auth') &&
    !request.nextUrl.pathname.startsWith('/student/auth')
  ) {
    // no user, redirect to login page
    const url = request.nextUrl.clone()
    url.pathname = '/student/auth/login'
    return NextResponse.redirect(url)
  }

  return supabaseResponse
}
```

Listing 13: Authentication Middleware

5.8 Real-time Features

The platform implements several real-time features to enhance user experience and engagement.

5.8.1 Stream Video Provider

The Stream Video Provider component enables real-time video calls throughout the application.

```
"use client";

import { ReactNode, useEffect, useState } from "react";
import {
  StreamVideo,
  StreamVideoClient,
  User,
} from "@stream-io/video-react-sdk";
import { createToken } from "@/actions/stream";

interface StreamVideoProviderProps {
  children: ReactNode;
  user: User;
}

export default function StreamVideoProvider({
  children,
  user,
}: StreamVideoProviderProps) {
  const [client, setClient] = useState<StreamVideoClient | null>(null);
  const apiKey = process.env.NEXT_PUBLIC_STREAM_API_KEY!;

  useEffect(() => {
    const initializeClient = async () => {
      try {
        const token = await createToken();
        const client = new StreamVideoClient({
          apiKey,
          user,
          token,
        });

        await client.connectUser();
        setClient(client);
      } catch (error) {
        console.error("Error connecting to Stream:", error);
      }
    };

    if (!client && user.id) {
      initializeClient();
    }
```

```
    return () => {
      if (client) {
        client.disconnectUser();
        setClient(null);
      }
    };
  }, [apiKey, client, user]);

  if (!client) {
    return <div>Loading video client...</div>;
  }

  return <StreamVideo client={client}>{children}</StreamVideo>;
}
```

Listing 14: Stream Video Provider

5.8.2 Stream Chat Component

The Stream Chat component provides real-time messaging capabilities.

```
"use client";

import { useEffect, useState } from "react";
import { StreamChat } from "stream-chat";
import {
  Chat,
  Channel,
  ChannelHeader,
  MessageInput,
  MessageList,
  Thread,
  Window,
} from "stream-chat-react";
import { tokenProvider } from "@/actions/stream";
import "stream-chat-react/dist/css/index.css";

interface StreamChatProps {
  user: {
    id: string;
    name: string;
    image?: string;
  };
  channelId: string;
}

export default function StreamChat({ user, channelId }: StreamChatProps) {
  const [client, setClient] = useState<StreamChat | null>(null);
  const [channel, setChannel] = useState<any>(null);

  useEffect(() => {
    const initChat = async () => {
      try {
        const token = await tokenProvider();
        const chatClient = StreamChat.getInstance(
          process.env.NEXT_PUBLIC_STREAM_API_KEY!
        );

        await chatClient.connectUser(
          {
            id: user.id,
            name: user.name,
            image: user.image,
          },
          token
        );

        const channel = chatClient.channel("messaging", channelId);
        await channel.watch();
```

```
        setClient(chatClient);
        setChannel(channel);
      } catch (error) {
        console.error("Error initializing chat:", error);
      }
    };

    initChat();

    return () => {
      if (client) {
        client.disconnectUser();
      }
    };
  }, [channelId, user]);

  if (!client || !channel) {
    return <div>Loading chat...</div>;
  }

  return (
    <div className="h-[calc(100vh-64px)]">
      <Chat client={client} theme="messaging light">
        <Channel channel={channel}>
          <Window>
            <ChannelHeader />
            <MessageList />
            <MessageInput />
          </Window>
          <Thread />
        </Channel>
      </Chat>
    </div>
  );
}
```

Listing 15: Stream Chat Component

5.9 Data Management

The application implements efficient data management patterns to handle user data and content.

5.9.1 Follow/Unfollow Functionality

The follow/unfollow system manages relationships between students and instructors.

```
export const handleFollow = async ({
  id,
  followers,
  userID,
}: {
  id: string;
  userID: string;
  followers: string[];
}) => {
  const supabase = createClient();
  const { data, error } = await supabase
    .from("students")
    .select("following_list")
    .eq("id", userID);
  if (error || !data) {
    return false;
  }

  const studentFollowingList = data[0].following_list;
  const newStudentFollowing = [...studentFollowingList, id];
  const newInstructorFollowers = [...followers, userID];
```

```
  const { error: studentError } = await supabase
    .from("students")
    .update({ following_list: newStudentFollowing })
    .eq("id", userID);

  if (studentError) {
    return false;
  }

  const { error: instructorError } = await supabase
    .from("instructors")
    .update({ followers: newInstructorFollowers })
    .eq("id", id);

  if (instructorError) {
    return false;
  }
  return true;
};

export const handleUnFollow = async (
  instructorFollowers: string[],
  userID: string,
  instructorID: string
) => {
  const supabase = createClient();
  const { data, error } = await supabase
    .from("students")
    .select("following_list")
    .eq("id", userID);
  if (error || !data) {
    return false;
  }
  const studentFollowingList = data[0].following_list;
  const newStudentFollowing = studentFollowingList.filter(
    (following: string) => following !== instructorID
  );
  const newInstructorFollowers = instructorFollowers.filter(
    (follower) => follower !== userID
  );

  const { error: studentError } = await supabase
    .from("students")
    .update({ following_list: newStudentFollowing })
    .eq("id", userID);
  if (studentError) {
    return false;
  }
  const { error: instructorError } = await supabase
    .from("instructors")
    .update({ followers: newInstructorFollowers })
    .eq("id", instructorID);

  if (instructorError) {
    return false;
  }
  return true;
};
```

Listing 16: Follow/Unfollow Functions

5.9.2 Announcement Management

Instructors can create and manage announcements that are displayed in student feeds.

```
"use client";

import { useState } from "react";
```

```
import { useRouter } from "next/navigation";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { useToast } from "@/hooks/use-toast";
import { createClient } from "@/lib/supabase/client";

interface AnnouncementFormProps {
  user: {
    id: string;
    name: string;
    image: string;
    occupation: string;
    interest: string;
  };
}

export default function AnnouncementForm({ user }: AnnouncementFormProps) {
  const [content, setContent] = useState("");
  const [isLoading, setIsLoading] = useState(false);
  const { toast } = useToast();
  const router = useRouter();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setIsLoading(true);

    try {
      const supabase = createClient();
      const { error } = await supabase.from("announcements").insert({
        author_id: user.id,
        author_name: user.name,
        author_image: user.image,
        author_title: user.occupation,
        content,
        interest: user.interest,
        likes: [],
      });

      if (error) {
        throw error;
      }

      toast({
        title: "Announcement created",
        description: "Your announcement has been published successfully.",
      });

      setContent("");
      router.refresh();
    } catch (error) {
      console.error("Error creating announcement:", error);
      toast({
        title: "Error",
        description: "Failed to create announcement. Please try again.",
        variant: "destructive",
      });
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-4 p-4 bg-white rounded-lg shadow">
      <div className="space-y-2">
        <Label htmlFor="content">New Announcement</Label>
        <textarea
          id="content"
          value={content}
          onChange={(e) => setContent(e.target.value)}
```

```
            className="w-full min-h-[100px] p-2 border rounded-md"
            placeholder="Share an announcement with your followers..."
            required
          />
        </div>
        <Button type="submit" disabled={isLoading || !content.trim()}>
          {isLoading ? "Publishing..." : "Publish Announcement"}
        </Button>
      </form>
    );
}
```

Listing 17: Announcement Form Component

5.10 Responsive Design

The application is designed to work seamlessly across different device sizes.

```
"use client";

import { useState } from "react";
import Link from "next/link";
import { usePathname } from "next/navigation";
import { Button } from "@/components/ui/button";
import { Avatar, AvatarFallback, AvatarImage } from "@/components/ui/avatar";
import { logOut } from "@/actions/auth";
import { NavBarProps } from "@/types";
import { Menu, X, User, LogOut, Bell } from "lucide-react";

export default function NavBar({
  showFollowingList,
  setShowFollowingList,
  followingCount,
}: NavBarProps) {
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const pathname = usePathname();

  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  };

  const toggleFollowingList = () => {
    setShowFollowingList(!showFollowingList);
  };

  return (
    <nav className="bg-white shadow-sm">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="flex justify-between h-16">
          <div className="flex">
            <div className="flex-shrink-0 flex items-center">
              <Link href="/student/feed" className="text-xl font-bold text-primary">
                Sarvagya
              </Link>
            </div>
          </div>

          {/* Desktop navigation */}
          <div className="hidden sm:ml-6 sm:flex sm:items-center sm:space-x-4">
            <Button
              variant="ghost"
              onClick={toggleFollowingList}
              className="relative"
            >
              <Bell className="h-5 w-5 mr-1" />
              Following
              {followingCount > 0 && (
                <span className="absolute top-0 right-0 bg-primary text-white text-xs rounded-full h-5 w-5 flex items-center j
                  {followingCount}
```

```
                </span>
              )}
            </Button>

            <Button variant="ghost" onClick={() => logOut()}>
              <LogOut className="h-5 w-5 mr-1" />
              Logout
            </Button>

            <Avatar>
              <AvatarImage src="/placeholder-avatar.jpg" />
              <AvatarFallback>
                <User className="h-5 w-5" />
              </AvatarFallback>
            </Avatar>
          </div>

          {/* Mobile menu button */}
          <div className="flex items-center sm:hidden">
            <button
              onClick={toggleMenu}
              className="inline-flex items-center justify-center p-2 rounded-md text-gray-400 hover:text-gray-500 hover:bg-gra
            >
              <span className="sr-only">Open main menu</span>
              {isMenuOpen ? (
                <X className="block h-6 w-6" />
              ) : (
                <Menu className="block h-6 w-6" />
              )}
            </button>
          </div>
        </div>
      </div>

      {/* Mobile menu */}
      {isMenuOpen && (
        <div className="sm:hidden">
          <div className="pt-2 pb-3 space-y-1">
            <Button
              variant="ghost"
              onClick={toggleFollowingList}
              className="relative w-full justify-start"
            >
              <Bell className="h-5 w-5 mr-1" />
              Following
              {followingCount > 0 && (
                <span className="absolute top-2 left-24 bg-primary text-white text-xs rounded-full h-5 w-5 flex items-center j
                  {followingCount}
                </span>
              )}
            </Button>

            <Button
              variant="ghost"
              onClick={() => logOut()}
              className="w-full justify-start"
            >
              <LogOut className="h-5 w-5 mr-1" />
              Logout
            </Button>
          </div>
        </div>
      )}
    </nav>
  );
}
```

Listing 18: Responsive Navigation Bar

5.11 Performance Optimization

Several techniques are employed to optimize the application's performance.

5.11.1 Server Components

Next.js Server Components are used to reduce client-side JavaScript and improve initial load times.

```
import { Suspense } from "react";
import { createClient } from "@/lib/supabase/server";
import FeedComponent from "./(components)/FeedComponent";
import NavBar from "./(components)/NavBar";
import FollowingProfile from "./(components)/FollowingProfile";
import ContainerRelatedProfile from "./(components)/ContainerRelatedProfile";
import { getUserSession } from "@/actions/auth";

export default async function FeedPage() {
  const { user } = await getUserSession();
  const supabase = await createClient();

  // Fetch student data
  const { data: student } = await supabase
    .from("students")
    .select("*")
    .eq("id", user?.id)
    .single();

  // Fetch following list
  const { data: followingList } = await supabase
    .from("students")
    .select("following_list")
    .eq("id", user?.id)
    .single();

  const followingCount = followingList?.following_list?.length || 0;

  return (
    <div className="min-h-screen bg-gray-50">
      <Suspense fallback={<div>Loading navigation...</div>}>
        <NavBarWrapper followingCount={followingCount} />
      </Suspense>

      <main className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
        <div className="grid grid-cols-1 lg:grid-cols-12 gap-8">
          {/* Sidebar */}
          <aside className="lg:col-span-3">
            <Suspense fallback={<div>Loading profiles...</div>}>
              <FollowingWrapper userId={user?.id} />
            </Suspense>
          </aside>

          {/* Main content */}
          <div className="lg:col-span-6">
            <Suspense fallback={<div>Loading feed...</div>}>
              <FeedComponent userId={user?.id} />
            </Suspense>
          </div>

          {/* Related profiles */}
          <aside className="lg:col-span-3">
            <Suspense fallback={<div>Loading related profiles...</div>}>
              <ContainerRelatedProfile
                userID={user?.id}
                studentInterest={student?.interest}
              />
            </Suspense>
          </aside>
        </div>
      </main>
    </div>
```

```
  );
}

// Client components wrapped for state management
const NavBarWrapper = ({ followingCount }) => {
  "use client";
  const [showFollowingList, setShowFollowingList] = useState(false);

  return (
    <NavBar
      showFollowingList={showFollowingList}
      setShowFollowingList={setShowFollowingList}
      followingCount={followingCount}
    />
  );
};

const FollowingWrapper = ({ userId }) => {
  return <FollowingProfile userID={userId} />;
};
```

Listing 19: Server Component Example

5.11.2 Image Optimization

Next.js Image component is used to optimize image loading and delivery.

```
import Image from "next/image";
import { Avatar, AvatarFallback } from "@/components/ui/avatar";
import { getDatePosted } from "@/lib/utils";
import { Feed } from "@/types";

interface AnnouncementCardProps {
  announcement: Feed;
  onLike?: (id: number) => void;
  onDelete?: (id: string) => void;
  isOwner?: boolean;
  userId?: string;
}

export default function AnnouncementCard({
  announcement,
  onLike,
  onDelete,
  isOwner = false,
  userId,
}: AnnouncementCardProps) {
  const isLiked = userId ? announcement.likes.includes
(userId) : false;

  return (
    <div className="bg-white rounded-lg shadow p-4 mb-4">
      <div className="flex items-start space-x-3">
        <Avatar className="h-10 w-10">
          <Image
            src={announcement.author_image}
            alt={announcement.author_name}
            width={40}
            height={40}
            className="rounded-full object-cover"
            priority
          />
          <AvatarFallback>{announcement.author_name[0]}</AvatarFallback>
        </Avatar>

        <div className="flex-1">
          <div className="flex justify-between">
            <div>
              <h3 className="font-medium">{announcement.author_name}</h3>
```

```
            <p className="text-sm text-gray-500">{announcement.author_title}</p>
          </div>
          <span className="text-xs text-gray-500">
            {getDatePosted(announcement.created_at)}
          </span>
        </div>

        <p className="mt-2">{announcement.content}</p>

        <div className="mt-3 flex items-center justify-between">
          <button
            onClick={() => onLike && onLike(announcement.id)}
            className={`text-sm flex items-center ${
              isLiked ? "text-primary" : "text-gray-500"
            }`}
          >
            {announcement.likes.length} Likes
          </button>

          {isOwner && (
            <button
              onClick={() => onDelete && onDelete(announcement.id.toString())}
              className="text-sm text-red-500"
            >
              Delete
            </button>
          )}
        </div>
      </div>
    </div>
  );
}
```

Listing 20: Optimized Image Component

6. Testing

6.1 Testing Strategy

The Sarvagya platform employs a comprehensive testing strategy to ensure reliability, functionality, and performance across all components. The testing approach includes:

- **Unit Testing**: Individual components and functions are tested in isolation
- **Integration Testing**: Interactions between components are verified
- **End-to-End Testing**: Complete user flows are validated
- **Accessibility Testing**: Ensuring the platform is usable by all users

6.2 Testing Infrastructure

The testing infrastructure is built on Jest and React Testing Library, with custom utilities to support testing of Next.js components and server actions.

```
// jest.setup.js
import '@testing-library/jest-dom';

// Mock next/navigation
jest.mock('next/navigation', () => ({
  useRouter: jest.fn(() => ({
    push: jest.fn(),
    replace: jest.fn(),
    prefetch: jest.fn(),
    back: jest.fn(),
```

```
    forward: jest.fn(),
    refresh: jest.fn(),
    pathname: '/',
    query: {},
  })),
  usePathname: jest.fn(() => '/'),
  useSearchParams: jest.fn(() => new URLSearchParams()),
  useParams: jest.fn(() => ({})),
}));

// Mock Supabase
jest.mock('@supabase/ssr', () => ({
  createBrowserClient: jest.fn(() => ({
    auth: {
      signUp: jest.fn(),
      signInWithPassword: jest.fn(),
      signOut: jest.fn(),
      getUser: jest.fn(),
    },
    storage: {
      from: jest.fn(() => ({
        upload: jest.fn(),
      })),
    },
    from: jest.fn(() => ({
      select: jest.fn(() => ({
        eq: jest.fn(() => ({
          single: jest.fn(),
        })),
      })),
      insert: jest.fn(),
    })),
  })),
  createServerClient: jest.fn(),
}));
```

Listing 21: Test Setup File

6.3 Test Utilities

Custom test utilities are created to simplify testing of components with complex dependencies.

```
import React, { ReactElement } from 'react';
import { render, RenderOptions } from '@testing-library/react';

// Define a custom render function that includes providers
const customRender = (
  ui: ReactElement,
  options?: Omit<RenderOptions, 'wrapper'>,
) => {
  const AllProviders = ({ children }: { children: React.ReactNode }) => {
    return (
      <>
        {children}
      </>
    );
  };

  return render(ui, { wrapper: AllProviders, ...options });
};

// Re-export everything from testing-library
export * from '@testing-library/react';

// Override render method
export { customRender as render };
```

Listing 22: Test Utilities

6.4 Unit Tests

Unit tests verify the functionality of individual components and functions in isolation.

```
import { cn, getDatePosted, formatDateTime } from '../utils';

describe('cn function', () => {
  it('should merge class names correctly', () => {
    expect(cn('text-red-500', 'bg-blue-500')).toBe('text-red-500 bg-blue-500');
    expect(cn('p-4', { 'mt-2': true, 'mb-2': false })).toBe('p-4 mt-2');
    expect(cn('flex', ['items-center', 'justify-between'])).toBe('flex items-center justify-between');
  });
});

describe('getDatePosted function', () => {
  it('should return a formatted date string', () => {
    const now = new Date();
    const oneHourAgo = new Date(now.getTime() - 60 * 60 * 1000);
    const oneDayAgo = new Date(now.getTime() - 24 * 60 * 60 * 1000);

    expect(getDatePosted(oneHourAgo.toISOString())).toContain('hour ago');
    expect(getDatePosted(oneDayAgo.toISOString())).toContain('day ago');
  });
});
```

Listing 23: Utility Function Tests

6.5 Integration Tests

Integration tests verify that different components work together correctly.

```
import { getUserSession } from '../auth';
import { createClient } from '@/lib/supabase/server';

jest.mock('@/lib/supabase/server', () => ({
  createClient: jest.fn(),
}));

describe('Auth Actions', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  describe('getUserSession', () => {
    it('should return user data when session exists', async () => {
      const mockUser = { id: 'user-123', email: 'test@example.com' };
      const mockSupabase = {
        auth: {
          getUser: jest.fn().mockResolvedValue({
            data: { user: mockUser },
            error: null,
          }),
        },
      };

      (createClient as jest.Mock).mockResolvedValue(mockSupabase);

      const result = await getUserSession();

      expect(createClient).toHaveBeenCalled();
      expect(mockSupabase.auth.getUser).toHaveBeenCalled();
      expect(result).toEqual({
        error: null,
        status: 200,
        user: mockUser,
      });
    });
  });
});
```

Listing 24: Auth Action Tests

6.6 Test Coverage

The project maintains a high level of test coverage to ensure code quality and reliability.

```
# Run tests with coverage report
npm run test:coverage

# Output example:
# ----------------------|----------|----------|----------|----------|-------------------
# File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
# ----------------------|----------|----------|----------|----------|-------------------
# All files             |    85.62 |    76.19 |    83.33 |    85.62 |
#  src/actions          |    92.31 |    85.71 |    90.00 |    92.31 |
#  src/components       |    87.50 |    78.57 |    85.71 |    87.50 |
#  src/hooks            |    83.33 |    75.00 |    80.00 |    83.33 |
#  src/lib              |    90.91 |    83.33 |    88.89 |    90.91 |
# ----------------------|----------|----------|----------|----------|-------------------
```

Listing 25: Test Coverage Command

7. Future Work

7.1 Planned Enhancements

The Sarvagya platform has several planned enhancements to improve functionality and user experience:

- **Advanced Analytics**: Implement analytics dashboards for instructors to track student engagement and content performance
- **Content Recommendations**: Develop an AI-powered recommendation system to suggest relevant instructors and content to students
- **Interactive Learning Materials**: Add support for interactive learning materials such as quizzes, polls, and assignments
- **Mobile Applications**: Develop native mobile applications for iOS and Android to enhance the mobile experience
- **Offline Support**: Implement offline capabilities to allow users to access content without an internet connection

7.2 Technical Improvements

Several technical improvements are planned to enhance the platform's performance, scalability, and maintainability:

```
// Example of planned React Query implementation for data fetching
import { QueryClient, QueryClientProvider, useQuery } from 'react-query';

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 60 * 1000, // 1 minute
      cacheTime: 5 * 60 * 1000, // 5 minutes
      refetchOnWindowFocus: false,
      retry: 1,
    },
```

```
  },
});

function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <Dashboard />
    </QueryClientProvider>
  );
}

function Dashboard() {
  const { data, isLoading, error } = useQuery(
    'instructorStats',
    fetchInstructorStats,
    {
      onSuccess: (data) => {
        // Process data
      },
    }
  );

  // Component rendering
}
```

Listing 26: Planned Performance Optimization

7.3 Scalability Enhancements

To support growth in users and content, several scalability enhancements are planned:

- **Database Optimization**: Implement database indexing and query optimization for improved performance
- **Caching Strategy**: Develop a comprehensive caching strategy using Redis for frequently accessed data
- **Microservices Architecture**: Gradually migrate to a microservices architecture for better scalability and maintainability
- **Content Delivery Network**: Integrate with a CDN for faster delivery of static assets and content

7.4 Research Directions

Future research will focus on enhancing the educational experience through technology:

- **Learning Analytics**: Research on using learning analytics to improve educational outcomes
- **Personalized Learning Paths**: Develop algorithms for creating personalized learning paths based on student interests and performance
- **AI-Assisted Teaching**: Explore the use of AI to assist instructors in content creation and student engagement
- **Virtual Reality Integration**: Research the integration of VR for immersive learning experiences

8. Setup and Running Instructions

8.1 Prerequisites

Before setting up the Sarvagya platform, ensure you have the following prerequisites installed:

- Node.js (version 18.0.0 or higher)
- npm (version 8.0.0 or higher)
- Git
- Supabase account
- Stream account (for chat and video functionality)

8.2 Environment Setup

Set up the required environment variables for the application:

```
# Create a .env.local file in the project root
touch .env.local

# Add the following environment variables to the .env.local file
NEXT_PUBLIC_SUPABASE_URL=your_supabase_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key
STORAGE_URL=https://your_supabase_url/storage/v1/object/public/
NEXT_PUBLIC_STREAM_API_KEY=your_stream_api_key
STREAM_SECRET_KEY=your_stream_secret_key
NEXT_PUBLIC_PAGE_URL=http://localhost:3000
NEXT_PUBLIC_STREAM_CHANNEL_IMAGE_URL=https://api.dicebear.com/9.x/pixel-art/svg?seed=
```

Listing 27: Environment Variables Setup

8.3 Database Setup

Set up the required database tables and storage buckets in Supabase:

```
# Install Supabase CLI
npm install -g supabase

# Login to Supabase
supabase login

# Initialize Supabase project
supabase init

# Link to your Supabase project
supabase link --project-ref your_project_ref

# Apply migrations
supabase db push

# Create storage bucket for instructor headshots
supabase storage create headshots
```

Listing 28: Database Setup Commands

8.4 Installation

Install the project dependencies and set up the development environment:

```
# Clone the repository
git clone https://github.com/yourusername/sarvagya.git
cd sarvagya

# Install dependencies
npm install

# Run development server
npm run dev
```

Listing 29: Installation Commands

8.5 Running Tests

Run the test suite to ensure everything is working correctly:

```
# Run all tests
npm test

# Run tests in watch mode
npm run test:watch

# Run tests with coverage report
npm run test:coverage
```

Listing 30: Test Commands

8.6 Building for Production

Build and deploy the application for production:

```
# Build the application
npm run build

# Start the production server
npm start
```

Listing 31: Production Build Commands

8.7 Troubleshooting

Common issues and their solutions:

- **Authentication Issues**: Ensure your Supabase URL and anon key are correct in the .env.local file
- **Stream SDK Errors**: Verify that your Stream API key and secret are correctly set
- **Database Connection Issues**: Check that your Supabase project is active and accessible
- **Build Errors**: Clear the .next directory and node_modules, then reinstall dependencies

```
# Clear Next.js cache
rm -rf .next

# Reinstall dependencies
rm -rf node_modules
npm install

# Check environment variables
cat .env.local

# Verify Supabase connection
supabase status
```

Listing 32: Troubleshooting Commands

8.8 Deployment

Deploy the Sarvagya platform to a production environment:

```
# Deploy to Vercel
vercel

# Deploy to production
vercel --prod

# Set environment variables on Vercel
vercel env add NEXT_PUBLIC_SUPABASE_URL
vercel env add NEXT_PUBLIC_SUPABASE_ANON_KEY
vercel env add STORAGE_URL
vercel env add NEXT_PUBLIC_STREAM_API_KEY
vercel env add STREAM_SECRET_KEY
vercel env add NEXT_PUBLIC_PAGE_URL
vercel env add NEXT_PUBLIC_STREAM_CHANNEL_IMAGE_URL
```

Listing 33: Deployment Commands

8.9 Monitoring

Monitor the application in production to ensure optimal performance:

- Use Vercel Analytics to monitor page performance and user experience
- Set up Sentry for error tracking and monitoring
- Use Supabase dashboard to monitor database performance
- Configure Stream dashboard to monitor chat and video usage

```
// pages/_app.js
import { init } from '@sentry/nextjs';

init({
  dsn: process.env.SENTRY_DSN,
  tracesSampleRate: 1.0,
  environment: process.env.NODE_ENV,
});

function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />;
}

export default MyApp;
```

Listing 34: Sentry Integration

9. Conclusion

The Sarvagya platform represents a comprehensive solution for connecting students with instructors in a digital learning environment. By leveraging modern web technologies such as Next.js, Supabase, and Stream SDKs, the platform provides a seamless experience for real-time communication, content sharing, and relationship building.

The implementation follows best practices for code organization, testing, and security, ensuring a robust and maintainable codebase. The modular architecture allows for easy extension and enhancement, supporting the planned future work in analytics, personalization, and mobile applications.

The platform's focus on real-time interaction through chat and video calls, combined with the personalized content feed based on student interests and instructor relationships, creates a unique educational ecosystem that goes beyond traditional content delivery platforms.

As education continues to evolve in the digital age, Sarvagya aims to bridge the gap between knowledge seekers and knowledge providers, creating meaningful educational relationships that transcend geographical and socioeconomic barriers.

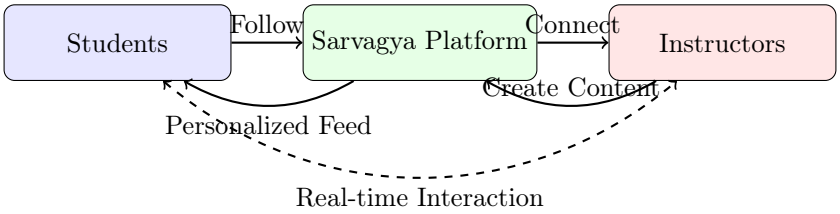

Figure 2: Sarvagya Platform Interaction Flow

Through continued development and research, Sarvagya aims to enhance the educational experience by providing tools that facilitate meaningful connections and knowledge exchange, ultimately contributing to a more accessible and personalized approach to learning in the digital age.

9.1 Conclusion

The Sarvagya platform implementation leverages modern web technologies to create a robust educational ecosystem. By combining Next.js for the frontend and backend, Supabase for authentication and data storage, and Stream SDKs for real-time communication, the platform provides a seamless experience for both students and instructors. The modular architecture and comprehensive test suite ensure maintainability and reliability, while security features like Row Level Security protect user data. The responsive design and performance optimizations make the platform accessible across various devices, creating an inclusive learning environment for all users.