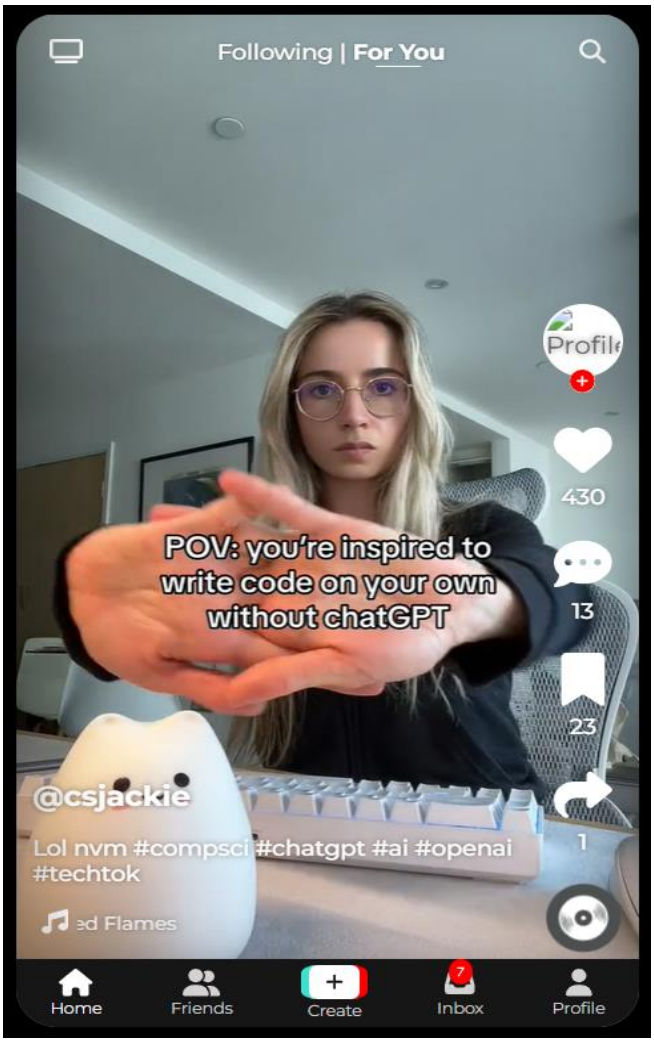
LAB 4: BUILD THE APPLICATION INTERFACE WITH REACTJS

Instruction section

In this exercise, we will build the interface of a TikTok-like application. This is the illustration for our application:



Please clone the project from GitHub using the following link:

https://github.com/thbinhh/faketiktok

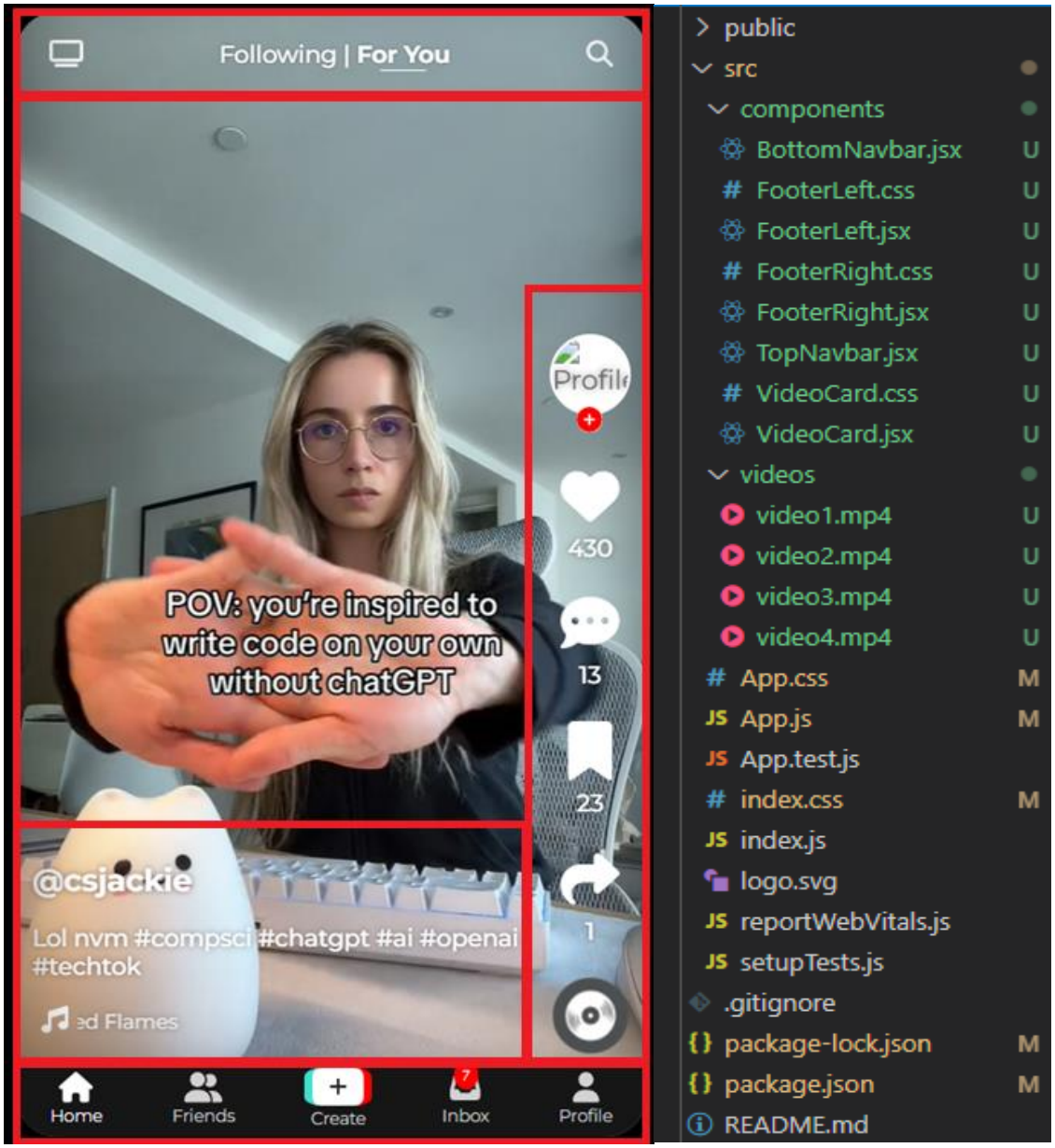
After cloning, run the installation commands below:

npm install

npm install --save @fortawesome/react-fontawesome

npm install --save @fortawesome/free-solid-svg-icons

We will divide the project into 5 main components as follows:



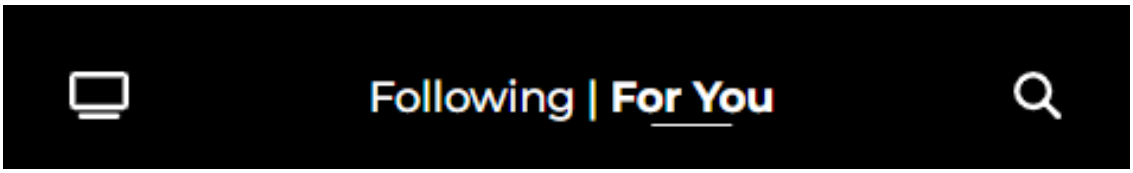
1. TopNavbar

```
import React from 'react';  6.9k (gzipped: 2.7k)
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faTv, faSearch } from '@fortawesome/free-solid-svg-icons';  885 (gzipped: 533)

const TopNavbar = () => {
  return (
    <div className="top-navbar">
      <FontAwesomeIcon icon={faTv} className='icon'/>
      <h2>Following  |   <span>For You</span></h2>
      <FontAwesomeIcon icon={faSearch} className='icon'/>
    </div>
  );
};

export default TopNavbar;
```

This code creates a top navigation bar for application:

- **TopNavbar Component**: A simple function that returns HTML-like JSX to describe the layout.
- **Icons**: Uses FontAwesome icons for a TV on the left and a search icon on the right.
- **Text in the Middle**: Shows "Following | For You" in the center, where "For You" is wrapped in a for separate styling.

Result:

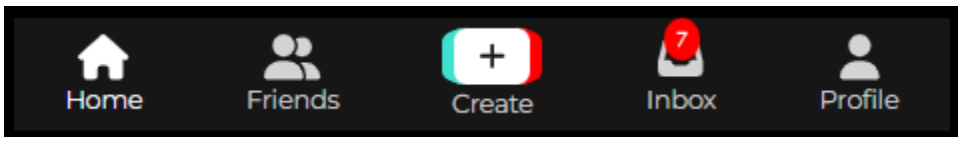


2. BottomNavbar:

```jsx
import React from 'react';  6.9k (gzipped: 2.7k)
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faHouse, faUserFriends, faPlus, faInbox, fa7,faUser } from '@fortawesome/free-solid-svg-icons';

function BottomNavbar() {
  return (
      <div className="bottom-navbar">
        <div className="nav-item">
          <FontAwesomeIcon icon={faHouse} className="icon active" />
          <span className="item-name active">Home</span>
        </div>
        <div className="nav-item">
          <FontAwesomeIcon icon={faUserFriends} className="icon" />
          <span className="item-name">Friends</span>
        </div>
        <div className="nav-item">
          <FontAwesomeIcon icon={faPlus} className="icon plus" />
          <span className="item-name">Create</span>
        </div>
        <div className="nav-item">
          <FontAwesomeIcon icon={fa7} className="notification" />
          <FontAwesomeIcon icon={faInbox} className="icon" />
          <span className="item-name">Inbox</span>
        </div>
        <div className="nav-item">
          <FontAwesomeIcon icon={faUser} className="icon" />
          <span className="item-name">Profile</span>
        </div>
      </div>
  );
}

export default BottomNavbar;
```

This **BottomNavbar** component creates a navigation bar for the bottom of application.

- **BottomNavbar Component**: A simple function that returns HTML-like JSX for layout.
- **Icons and Labels**: Each item has a FontAwesome icon and a label:
    - **Home** (active), **Friends**, **Create**, **Inbox** (with a notification icon), and **Profile**.
- **CSS Classes**: Classes like bottom-navbar, nav-item, icon, and item-name allow for styling.



3. VideoCard:

```
const VideoCard = (props) => {
  const { url, username, description, song, likes, shares, comments, saves, profilePic, setVideoRef, autoplay } = props;
  const videoRef = useRef(null);

  useEffect(() => {
    if (autoplay) {
      videoRef.current.play();
    }
  }, [autoplay]);

  const onVideoPress = () => {
    if (videoRef.current.paused) {
      videoRef.current.play();
    } else {
      videoRef.current.pause();
    }
  };
```

When we return, there will be two more components, **FooterLeft** and **FooterRight**, that will be implemented below**:**

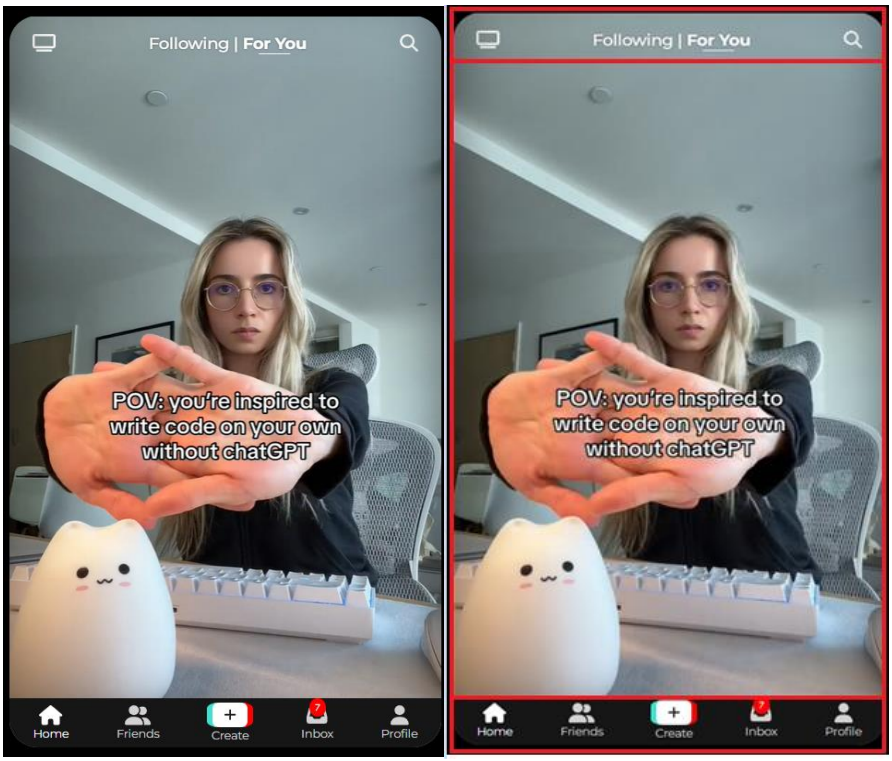
This **VideoCard** component creates a video player card with controls and details:

- **Props**: Receives details like url, username, description, song, likes, shares, comments, saves, profilePic, setVideoRef, and autoplay.
- **Video Control**:
    - Uses videoRef to control the video playback.
    - If autoplay is true, the video starts playing automatically when loaded.
- **Play/Pause on Click**:
    - The onVideoPress function toggles between play and pause when the video is clicked.

- **Returned JSX Structure**:

```jsx
  return (
    <div className="video">
      {/* The video element */}
      <video
        className="player"
        onClick={onVideoPress}
        ref={(ref) => {
          videoRef.current = ref;
          setVideoRef(ref);
        }}
        loop
        src={url}
      ></video>
      <div className="bottom-controls">
        <div className="footer-left">
          {/* The left part of the container */}

        </div>
        <div className="footer-right">
          {/* The right part of the container */}

        </div>
      </div>
    </div>
  );
};

export default VideoCard;
```

  - **<video> element**: Displays the video with click-to-play functionality, looping, and uses the url prop as the video source.
  - **Bottom Controls**: Divided into two parts:
    - **FooterLeft**: Displays details like username, description, and song.
    - **FooterRight**: Displays the user's profile picture and interaction stats (likes, shares, comments, saves).



4. FooterLeft:

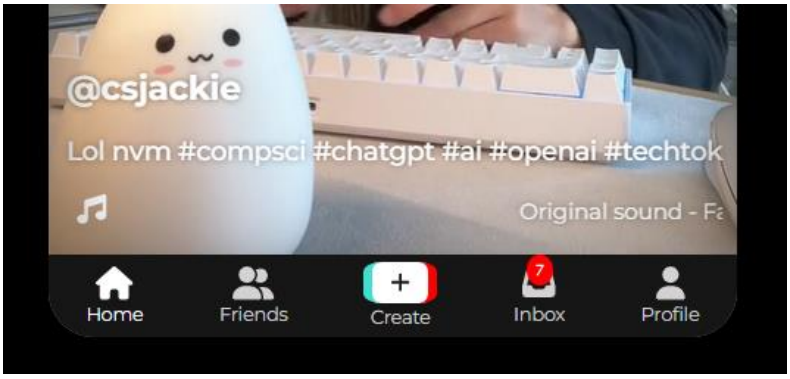
This **FooterLeft** component displays the left section of the footer on the video card, including the username, description, and song information:

1. **Props**:
    - Receives username, description, and song as props.
2. **Returned JSX Structure**:
    - **Outer Container (footer-container)**: Contains the footer layout and styling.
    - **Username and Description**:
        - The username is displayed as a heading (h3) with an @ symbol, e.g., @username.
        - The description is shown as a paragraph (p).
    - **Song Ticker**:
        - The song name is displayed with a music icon (faMusic from FontAwesome).

- A marquee element scrolls the song name from left to right, creating a ticker effect.

```
import React from 'react';  6.9k (gzipped: 2.7k)
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faMusic } from '@fortawesome/free-solid-svg-icons';  583 (gzipped: 370)
import './FooterLeft.css';

export default function FooterLeft(props) {
  const { username, description, song } = props;

  return (
    <div className="footer-container">
      <div className="footer-left">
        <div className="text">
          <h3>@{username}</h3>
          <p>{description}</p>
          <div className="ticker">
            <FontAwesomeIcon icon={faMusic} style={{ width: '30px' }} />
            {/* eslint-disable-next-line jsx-a11y/no-distracting-elements */}
            <marquee direction="left" scrollamount="2">
              <span>{song}</span>
            </marquee>
          </div>
        </div>
      </div>
    </div>
  );
}
```

Result:



5. FooterRight:

```
import React, { useState } from 'react';  6.9k (gzipped: 2.7k)
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faCirclePlus, faCircleCheck, faHeart, faCommentDots, faBookmark, faShare } from '@fortawesome/free-solid-svg-icons';
import './FooterRight.css';
```

```
function FooterRight({ likes, comments, saves, shares, profilePic }) {
  const [liked, setLiked] = useState(false);
  const [saved, setSaved] = useState(false);
  const [userAddIcon, setUserAddIcon] = useState(faCirclePlus);

  const handleUserAddClick = () => {
    setUserAddIcon(faCircleCheck);
    setTimeout(() => {
      setUserAddIcon(null);
    }, 3000); // Change the delay time (in milliseconds) as needed
  };

  // Function to convert likes count to a number
  const parseLikesCount = (count) => {
    if (typeof count === 'string') {
      if (count.endsWith('K')) {
        return parseFloat(count) * 1000;
      }
      return parseInt(count);
    }
    return count;
  };

  // Function to format likes count
  const formatLikesCount = (count) => {
    if (count >= 10000) {
      return (count / 1000).toFixed(1) + 'K';
    }
    return count;
  };

  const handleLikeClick = () => {
    setLiked((prevLiked) => !prevLiked);
  };
```

The **FooterRight** component adds interaction options on the video card, such as liking, saving, and following a user. Here's what it does:

1. **Props**:
    - Receives likes, comments, saves, shares, and profilePic as props.
2. **State Variables**:
    - liked: Tracks if the video is liked.
    - saved: Tracks if the video is saved.

- userAddIcon: Manages the icon for following/unfollowing a user.

3. **Functions**:
   - **handleUserAddClick**: Changes the follow icon to a check mark for a few seconds, then hides it.
   - **parseLikesCount**: Converts the likes prop from a shorthand (like "5K") to a number.
   - **formatLikesCount**: Formats the likes count to display in shorthand (e.g., 10000 becomes "10K").
   - **handleLikeClick**: Toggles the liked state when the like button is clicked.

When return **FooterRight** includes several different icons, such as:

```
return (
  <div className="footer-right">
```

```
<div className="sidebar-icon">
  {profilePic ? (
    // Displaying the user profile picture
    <img src={profilePic} className='userprofile' alt='Profile' style={{ width: '45px', height: '45px', color: '#616161' }} />
  ) : null}
  {/* The user add icon */}
  <FontAwesomeIcon icon={userAddIcon} className='useradd' style={{ width: '15px', height: '15px', color: '#FF0000' }}
  onClick={handleUserAddClick}/>
</div>
```

```
<div className="sidebar-icon">
  {/* The heart icon for liking */}
  <FontAwesomeIcon
    icon={faHeart}
    style={{ width: '35px', height: '35px', color: liked ? '#FF0000' : 'white' }}
    onClick={handleLikeClick}
  />
  {/* Displaying the formatted likes count */}
  <p>{formatLikesCount(parseLikesCount(likes) + (liked ? 1 : 0))}</p>
</div>
```

```
<div className="sidebar-icon">
  {/* The comment icon */}
  <FontAwesomeIcon icon={faCommentDots} style={{ width: '35px', height: '35px', color: 'white' }} />
  {/* Displaying the number of comments */}
  <p>{comments}</p>
</div>
```

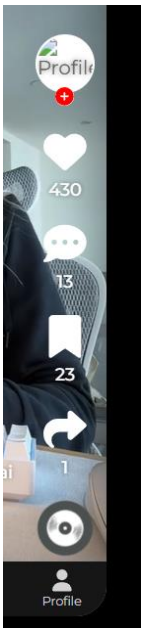
```jsx
<div className="sidebar-icon">
  {saved ? (
    // Displaying the bookmark icon when saved
    <FontAwesomeIcon
      icon={faBookmark}
      style={{ width: '35px', height: '35px', color: '#ffc107' }}
      onClick={() => setSaved(false)}
    />
  ) : (
    // Displaying the bookmark icon when not saved
    <FontAwesomeIcon
      icon={faBookmark}
      style={{ width: '35px', height: '35px', color: 'white' }}
      onClick={() => setSaved(true)}
    />
  )}
  {/* Displaying the number of saves */}
  <p>{saved ? saves + 1 : saves}</p>
</div>
```

```jsx
<div className="sidebar-icon">
  {/* The share icon */}
  <FontAwesomeIcon icon={faShare} style={{ width: '35px', height: '35px', color: 'white' }} />
  {/* Displaying the number of shares */}
  <p>{shares}</p>
</div>
```

```jsx
<div className="sidebar-icon record">
  {/* Displaying the record icon */}
  <img src="https://static.thenounproject.com/png/934821-200.png" alt='Record Icon' />
</div>
```

```jsx
return (
  <div className="footer-right">
    <div className="sidebar-icon">···
    </div>

    <div className="sidebar-icon">···
    </div>

    <div className="sidebar-icon">···
    </div>

    <div className="sidebar-icon">···
    </div>

    <div className="sidebar-icon">···
    </div>

    <div className="sidebar-icon record">···
    </div>
  </div>
```

Result:



Complete the **VideoCard** section:

```
  return (
    <div className="video">
      {/* The video element */}
      <video
        className="player"
        onClick={onVideoPress}
        ref={(ref) => {
          videoRef.current = ref;
          setVideoRef(ref);
        }}
        loop
        src={url}
      ></video>
      <div className="bottom-controls">
        <div className="footer-left">
          {/* The left part of the container */}
          <FooterLeft username={username} description={description} song={song} />
        </div>
        <div className="footer-right">
          {/* The right part of the container */}
          <FooterRight likes={likes} shares={shares} comments={comments} saves={saves} profilePic={profilePic} />
        </div>
      </div>
    </div>
  );
};

export default VideoCard;
```

6. App.js file:

We will create an array called **videosUrls**

```
const videoUrls = [
  {
    url: require('./videos/video1.mp4'),
    profilePic: 'https://p16-sign-useast2a.tiktokcdn.com/tos-useast2a-avt-0068-giso/9d429ac49d6d18de6ebd2a3fb1f39269~c5_10
    username: 'csjackie',
    description: 'Lol nvm #compsci #chatgpt #ai #openai #techtok',
    song: 'Original sound - Famed Flames',
    likes: 430,
    comments: 13,
    saves: 23,
    shares: 1,
  },
  {
    url: require('./videos/video2.mp4'),
    profilePic: 'https://p16-sign-va.tiktokcdn.com/tos-maliva-avt-0068/eace3ee69abac57c39178451800db9d5~c5_100x100.jpeg?x-
    username: 'dailydotdev',
    description: 'Every developer brain @francesco.ciulla #developerjokes #programming #programminghumor #programmingmemes
    song: 'tarawarolin wants you to know this isnt my sound - Chaplain J Rob',
    likes: '13.4K',
    comments: 3121,
    saves: 254,
    shares: 420,
  },
```

This code defines an array, videoUrls, containing objects with details for each video. Here's a breakdown of each part:

1. **Array of Video Objects**:
    - Each object represents a video with details about the video file, user, and stats.

2. **Properties for Each Video**:
    - **url**: The file path for the video (e.g., video1.mp4, video2.mp4), using require() to load it from the local project.
    - **profilePic**: A URL for the user's profile picture, linking to an external image.
    - **username**: The username of the content creator (e.g., csjackie, dailydotdev).
    - **description**: A short description of the video, often with hashtags.
    - **song**: The song or audio used in the video, credited to the original creator.
    - **likes, comments, saves, shares**: Statistics for each video:
        - **likes**: Can be a number (e.g., 430) or shorthand (e.g., "13.4K").
        - **comments**: Total comment count.
        - **saves**: Number of times the video was saved.
        - **shares**: Number of times the video was shared.

Handle **play/pause** for the videos:

```
function App() {
  const [videos, setVideos] = useState([]);
  const videoRefs = useRef([]);

  useEffect(() => {
    setVideos(videoUrls);
  }, []);

  useEffect(() => {
    const observerOptions = {
      root: null,
      rootMargin: '0px',
      threshold: 0.8, // Adjust this value to change the scroll trigger point
    };

    // This function handles the intersection of videos
    const handleIntersection = (entries) => {
      entries.forEach((entry) => {
        if (entry.isIntersecting) {
          const videoElement = entry.target;
          videoElement.play();
        } else {
          const videoElement = entry.target;
          videoElement.pause();
        }
      });
    };
```

1. **State and Refs**:

   videos: State that stores video data from videoUrls.

   videoRefs: A ref array to keep references to each video element for controlling play and pause.

2. **useEffect to Set Videos**:

   Loads **videoUrls** into videos state when the component mounts.

```
  const observer = new IntersectionObserver(handleIntersection, observerOptions);

  // We observe each video reference to trigger play/pause
  videoRefs.current.forEach((videoRef) => {
    observer.observe(videoRef);
  });

  // We disconnect the observer when the component is unmounted
  return () => {
    observer.disconnect();
  };
}, [videos]);

// This function handles the reference of each video
const handleVideoRef = (index) => (ref) => {
  videoRefs.current[index] = ref;
};
```

3. **Intersection Observer**:

   Sets up an IntersectionObserver to watch each video. When a video is at least 80% visible (as set by threshold: 0.8), it will automatically play. Otherwise, it pauses.

   Observes each video in videoRefs and disconnects the observer when the component unmounts.

Map data to **VideoCard**:

```
  return (
    <div className="app">
      <div className="container">
        <TopNavbar className="top-navbar" />
        {/* Here we map over the videos array and create VideoCard components */}
        {videos.map((video, index) => (
          <VideoCard
            key={index}
            username={video.username}
            description={video.description}
            song={video.song}
            likes={video.likes}
            saves={video.saves}
            comments={video.comments}
            shares={video.shares}
            url={video.url}
            profilePic={video.profilePic}
            setVideoRef={handleVideoRef(index)}
            autoplay={index === 0}
          />
        ))}
        <BottomNavbar className="bottom-navbar" />
      </div>
    </div>
  );

}

export default App;
```

Rendering the Components:

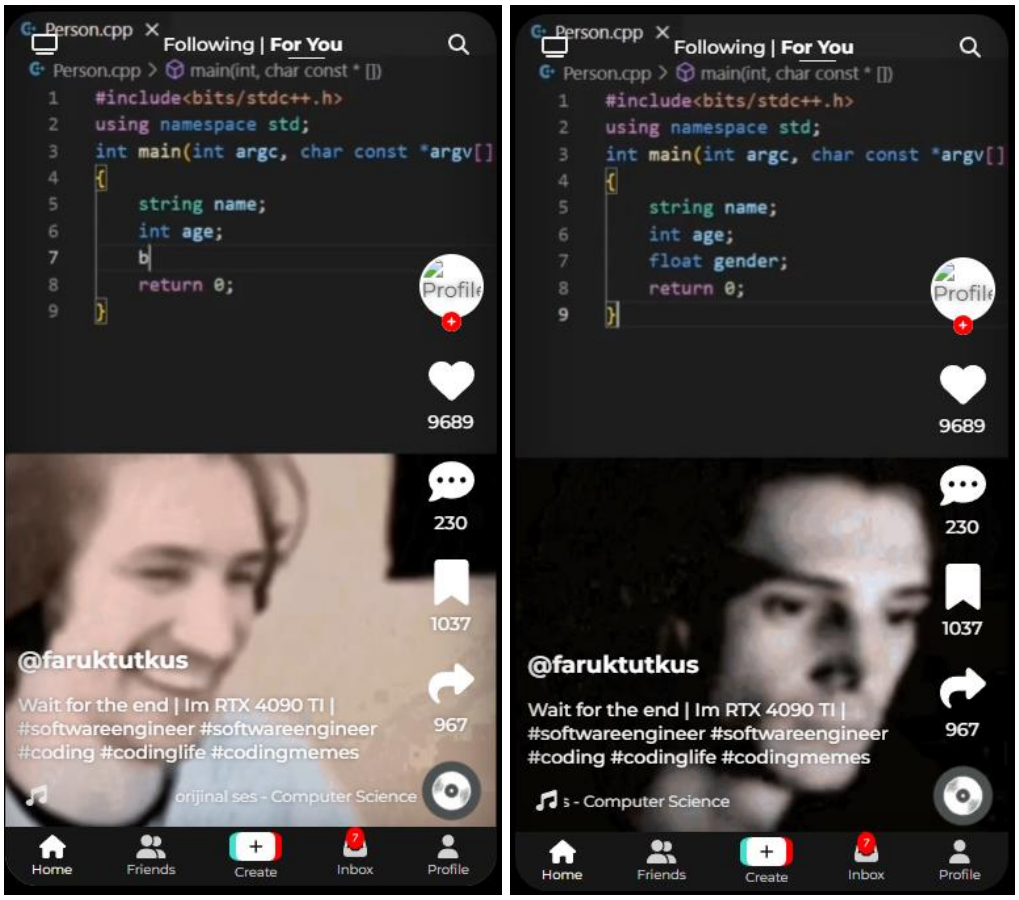
Top Navbar: Displays a navigation bar at the top using <TopNavbar />.

Video Cards: Maps over the videos array and renders a VideoCard for each video, passing in:

- Video data (e.g., username, description, song, likes).
- The ref for each video via **setVideoRef**.
- **autoplay={index === 0}**, so only the first video auto-plays on load.

Bottom Navbar: Displays a navigation bar at the bottom using <BottomNavbar />.

The completed application will look like this:



Exercise section:

1. Change the profile pictures of the users.
2. Add a button in FooterRight to mute and unmute the video when clicked.
3. Our application currently cannot holding the mouse and moving up or down navigates to the next or previous video, add this functionality.
4. When selecting the 'Save' button, automatically copy the video URL.
5. Create a custom User interface. When scrolling or pressing the right arrow key, it should display the video upload information (similar to TikTok).
6. When selecting the 'Share' button, a popup will appear with options to share on Facebook, Instagram, Thread (only visible). Press the X button to close it. *
7. Create a search function: When clicking the magnifying glass icon, entering a hashtag, and pressing Enter, the app will only display videos with that hashtag. *

The submission includes a **GitHub link** and a **PDF file** detailing the code changes, with illustrative images of the results (except for question 2, 3).

"Good luck with this lab assignment!"