



## Code Document

### Movie Ticket Reservation

จัดทำโดย

นางสาว กิตติมา	แพงไพรี	รหัสนักศึกษา 6504062630022
นาย ณัฐนันท์	พাঠกากลุจน์	รหัสนักศึกษา 6504062630111
นาย ชนิตศักดิ์	สกุลรัศมีหรรษ	รหัสนักศึกษา 6504062630138
นาย พงศธร	รอดเงิน	รหัสนักศึกษา 6504062630162
นาย มโนนัท	ธนาโชติพงศา	รหัสนักศึกษา 6504062630227
นาย วชิรสรณ์	เติมรัตนสุวรรณ	รหัสนักศึกษา 6504062630251

อาจารย์ผู้สอน  
อาจารย์สกิตย์ ประสมพันธ์

โครงการนี้เป็นส่วนหนึ่งของรายวิชา Software engineering  
ภาคเรียนที่ 2/2566

ภาควิชาคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์  
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

## สารบัญ

ชื่อไฟล์	หน้า
<b>การเรียกใช้งาน API (Calling APIs) .....</b>	<b>1</b>
booking.js.....	1
index.js.....	4
movies.js.....	5
theatres.js.....	8
users.js .....	17
<b>ส่วนประกอบ (Components) .....</b>	<b>19</b>
Button.js .....	19
PageTitle.js .....	20
ProtectedRoute.js .....	21
<b>ส่วนผู้ดูแลระบบ (Admin Section) .....</b>	<b>24</b>
index.js.....	24
MovieForm.js .....	25
MoviesList.js.....	29
TheatresList.js .....	34
<b>หน้าจองจอดที่นั่ง (Seat Reservation Page) .....</b>	<b>38</b>
<b>หน้าหลัก (Home Page).....</b>	<b>45</b>
<b>หน้าเข้าสู่ระบบ (Login Page) .....</b>	<b>47</b>
<b>โปรไฟล์ (Profile) .....</b>	<b>49</b>
Bookings.js .....	49
index.js.....	51
<b>สมัครใช้งาน (Register) .....</b>	<b>52</b>

ชื่อไฟล์	หน้า
รายการการแสดง (Show Listings).....	54
สถานที่แสดง (Theatres) .....	60
index.js.....	60
TheatreForm.js .....	61
TheatresList.js .....	64
โรงภาพยนตร์สำหรับฉาย (Theatres for Movie).....	69
การจัดการสถานะด้วย Redux (State Management with Redux) .....	73
loadersSlice.js.....	73
store.js.....	74
usersSlice.js.....	75
การกำหนดค่าเซิร์ฟเวอร์ (Server Configuration) .....	76
dbConfig.js.....	76
server.js.....	77
มิดเดิลแวร์ (Middlewares) .....	78
เส้นทางเซิร์ฟเวอร์ (Server Routes) .....	79
bookingsRoute.js .....	79
moviesRoute.js .....	83
theatresRoute.js .....	88
usersRoute.js.....	98

## การเรียกใช้งาน API (Calling APIs)

### booking.js (1/3)

**คำอธิบาย :** ไฟล์นี้เป็นส่วนหนึ่งของโปรแกรมที่เกี่ยวข้องกับการจองตั๋วการแสดงหรือบริการที่เกี่ยวข้องกับการแสดงของบริษัทหรือเว็บไซต์นึง โดยมีหน้าที่หลักๆ คือการสื่อสารกับเซิร์ฟเวอร์ผ่าน API เพื่อทำการชำระเงิน จองตั๋วการแสดง และรับข้อมูลการจองของผู้ใช้ โดยมีรายละเอียดดังนี้

1. MakePayment: ฟังก์ชันนี้ใช้สำหรับการทำการชำระเงิน โดยรับโทเค็นของผู้ที่ต้องการชำระเงิน และจำนวนเงินที่ต้องการชำระเข้ามา เมื่อเรียกใช้งาน ฟังก์ชันนี้จะทำการส่งคำขอ HTTP POST ไปยัง เส้นทาง '/api/bookings/make-payment' บนเซิร์ฟเวอร์ และคืนข้อมูลการชำระเงินที่ได้รับจากเซิร์ฟเวอร์ เซิร์ฟเวอร์กลับมา



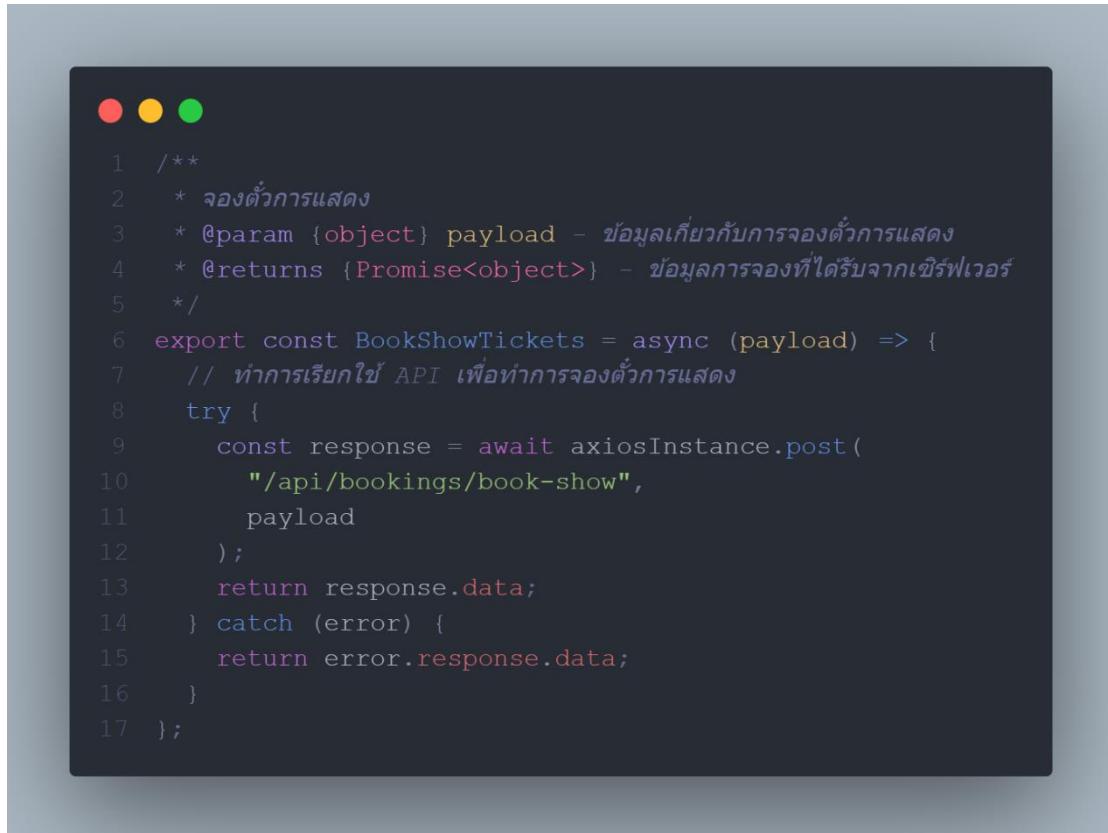
```

1  /**
2   * ทำการชำระเงิน
3   * @param {string} token - โทเค็นของผู้ที่ทำการชำระเงิน
4   * @param {number} amount - จำนวนเงินที่ต้องการชำระ
5   * @returns {Promise<object>} - ข้อมูลการชำระเงินที่ได้รับจากเซิร์ฟเวอร์
6  */
7  export const MakePayment = async (token, amount) => {
8    // ทำการเรียกใช้ API เพื่อทำการชำระเงิน
9    try {
10      const response = await axiosInstance.post("/api/bookings/make-payment", {
11        token,
12        amount,
13      });
14      return response.data;
15    } catch (error) {
16      return error.response.data;
17    }
18  };

```

## booking.js (2/3)

2.BookShowTickets: พังก์ชันนี้ใช้สำหรับการจองตั๋วการแสดง โดยรับข้อมูลเกี่ยวกับการจองตั๋วการแสดงเข้ามา เมื่อเรียกใช้งาน พังก์ชันนี้จะทำการส่งคำขอ HTTP POST ไปยังเส้นทาง '/api/bookings/book-show' บนเซิร์ฟเวอร์ และคืนข้อมูลการจองที่ได้รับจากเซิร์ฟเวอร์กลับมา

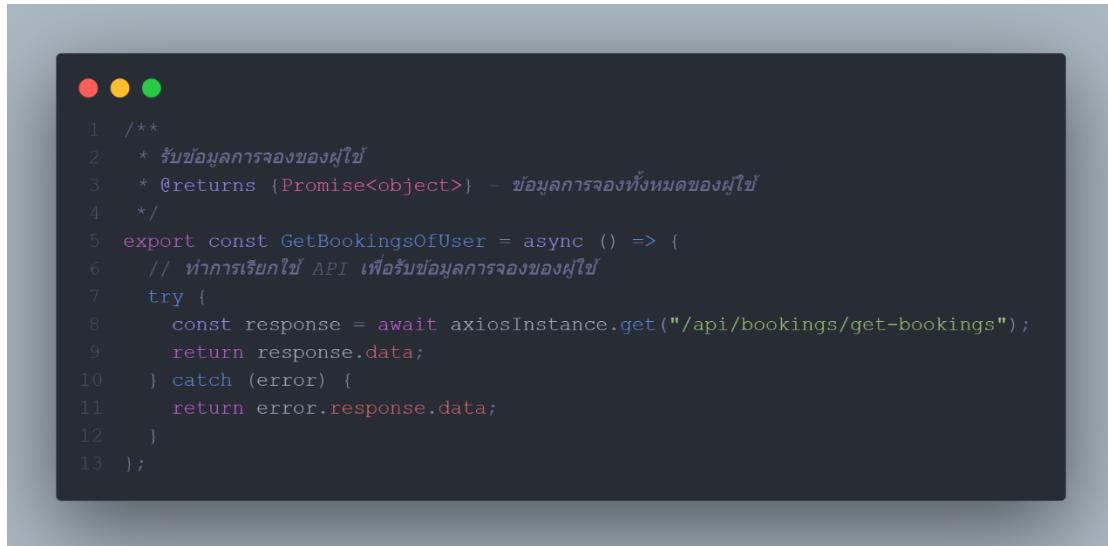


```
1  /**
2   * จองตั๋วการแสดง
3   * @param {object} payload - ข้อมูลเกี่ยวกับการจองตั๋วการแสดง
4   * @returns {Promise<object>} - ข้อมูลการจองที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const BookShowTickets = async (payload) => {
7    // ทำการเรียกใช้ API เพื่อทำการจองตั๋วการแสดง
8    try {
9      const response = await axiosInstance.post(
10        "/api/bookings/book-show",
11        payload
12      );
13      return response.data;
14    } catch (error) {
15      return error.response.data;
16    }
17  };

```

### booking.js (3/3)

3.GetBookingsOfUser: พังก์ชันนี้ใช้สำหรับการรับข้อมูลการจองของผู้ใช้ โดยไม่ต้องรับพารามิเตอร์เข้ามา เมื่อเรียกใช้งาน พังก์ชันนี้จะทำการส่งคำขอ HTTP GET ไปยังเส้นทาง '/api/bookings/get-bookings' บนเซิร์ฟเวอร์ และคืนข้อมูลการจองทั้งหมดของผู้ใช้กลับมา



```
1  /**
2   * ดูบข้อมูลการจองของผู้ใช้
3   * @returns {Promise<object>} - ข้อมูลการจองทั้งหมดของผู้ใช้
4  */
5  export const GetBookingsOfUser = async () => {
6    // ทำการเรียกใช้ API เพื่อรับข้อมูลการจองของผู้ใช้
7    try {
8      const response = await axiosInstance.get("/api/bookings/get-bookings");
9      return response.data;
10    } catch (error) {
11      return error.response.data;
12    }
13  };
```

## index.js

คำอธิบาย : ไฟล์นี้เป็นการสร้าง Axios instance ซึ่งเป็นอ็อบเจกต์ที่กำหนดการตั้งค่าเฉพาะสำหรับการส่งคำขอ HTTP ไปยังเซิร์ฟเวอร์ โดยมีการตั้งค่า header ให้กับทุกคำขอโดยอัตโนมัติ โดยใช้โทเค็นที่เก็บใน localStorage เพื่อทำการยืนยันตัวตน ในรูปแบบของ Bearer token ที่ส่งผ่าน authorization header เมื่อสร้าง Axios instance นี้แล้ว คุณสามารถนำมายังงาน API ได้ทันที โดยที่ไม่ต้องกำหนด header แบบเดิมในทุกคำขออีกต่อไป



```
1 import axios from "axios";
2
3 /**
4  * Axios instance ที่ใช้สำหรับการส่งคำขอ HTTP ไปยังเซิร์ฟเวอร์
5  * โดยมีการเซ็ต header ให้กับทุกคำขอโดยอัตโนมัติ
6  * โดยใช้โทเค็นที่เก็บใน localStorage เพื่อทำการยืนยันตัวตน
7  */
8 export const axiosInstance = axios.create({
9   headers: {
10     "Content-Type": "application/json",
11     authorization: `Bearer ${localStorage.getItem("token")}`,
12   },
13 });
14
```

### movies.js (1/3)

**คำอธิบาย :** ไฟล์นี้เป็นส่วนของโปรแกรมที่เกี่ยวกับการจัดการข้อมูลหนัง โดยมีฟังก์ชันที่เรียกใช้ Axios instance เพื่อส่งคำขอ API ไปยังเซิร์ฟเวอร์ ซึ่งมีรายละเอียดดังนี้

1.AddMovie: ฟังก์ชันนี้ใช้สำหรับเพิ่มข้อมูลหนังใหม่ เมื่อเรียกใช้งาน ฟังก์ชันนี้จะทำการส่งคำขอ HTTP POST ไปยังเส้นทาง '/api/movies/add-movie' บนเซิร์ฟเวอร์ พร้อมกับส่งข้อมูลเกี่ยวกับหนังที่ต้องการเพิ่มเข้าไป และจะคืนข้อมูลที่ได้รับจากเซิร์ฟเวอร์กลับมา



```

1  /**
2   * เพิ่มหนังใหม่
3   * @param {object} payload - ข้อมูลเกี่ยวกับหนังที่ต้องการเพิ่ม
4   * @returns {Promise<object>} - ข้อมูลการเพิ่มหนังที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const AddMovie = async (payload) => {
7   try {
8     const response = await axiosInstance.post("/api/movies/add-movie", payload);
9     return response.data;
10  } catch (error) {
11    return error.response;
12  }
13 };

```

2.GetAllMovies: ฟังก์ชันนี้ใช้สำหรับรับข้อมูลรายการหนังทั้งหมด เมื่อเรียกใช้งาน ฟังก์ชันนี้จะทำการส่งคำขอ HTTP GET ไปยังเส้นทาง '/api/movies/get-all-movies' บนเซิร์ฟเวอร์ และคืนข้อมูลรายการหนังทั้งหมดที่ได้รับจากเซิร์ฟเวอร์กลับมา



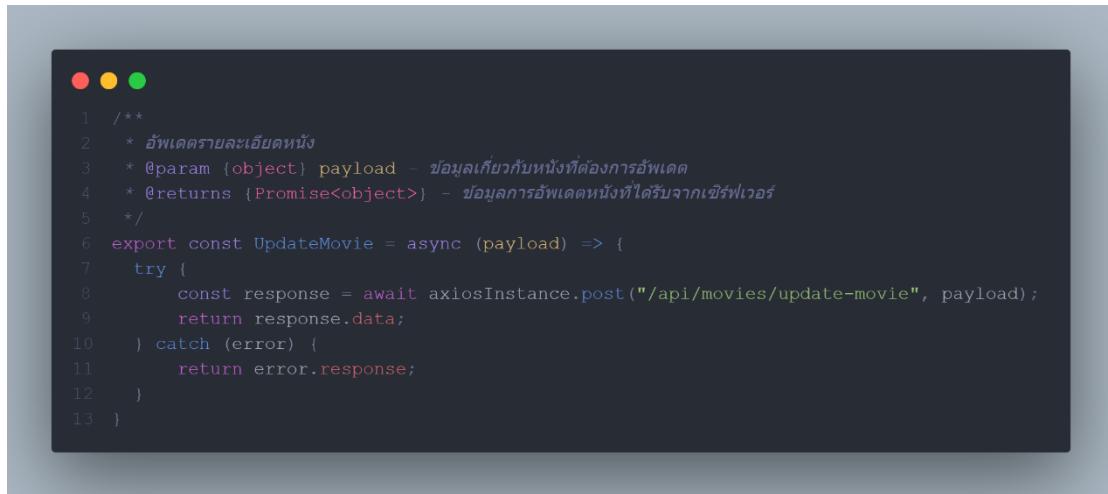
```

1  /**
2   * แสดงรายการหนังทั้งหมด
3   * @returns {Promise<object>} - ข้อมูลรายการหนังทั้งหมดที่ได้รับจากเซิร์ฟเวอร์
4  */
5 export const GetAllMovies = async () => {
6   try {
7     const response = await axiosInstance.get("/api/movies/get-all-movies");
8     return response.data;
9   } catch (error) {
10    return error.response;
11  }
12 };

```

## movies.js (2/3)

3.UpdateMovie: พังก์ชันนี้ใช้สำหรับอัปเดตรายละเอียดของหนัง เมื่อเรียกใช้งาน พังก์ชันนี้จะทำการส่งคำขอ HTTP POST ไปยังเส้นทาง '/api/movies/update-movie' บนเซิร์ฟเวอร์ พร้อมกับส่งข้อมูลเกี่ยวกับหนังที่ต้องการอัปเดตเข้าไป และจะคืนข้อมูลการอัปเดตหนังที่ได้รับจากเซิร์ฟเวอร์



```

1  /**
2  * อัปเดตรายละเอียดหนัง
3  * @param {object} payload - ข้อมูลเกี่ยวกับหนังที่ต้องการอัปเดต
4  * @returns {Promise<object>} - ข้อมูลการอัปเดตหนังที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const UpdateMovie = async (payload) => {
7   try {
8     const response = await axiosInstance.post("/api/movies/update-movie", payload);
9     return response.data;
10  } catch (error) {
11    return error.response;
12  }
13}

```

4.DeleteMovie: พังก์ชันนี้ใช้สำหรับลบรายการหนัง เมื่อเรียกใช้งาน พังก์ชันนี้จะทำการส่งคำขอ HTTP POST ไปยังเส้นทาง '/api/movies/delete-movie' บนเซิร์ฟเวอร์ พร้อมกับส่งข้อมูลเกี่ยวกับหนังที่ต้องการลบเข้าไป และจะคืนข้อมูลการลบหนังที่ได้รับจากเซิร์ฟเวอร์กลับมา



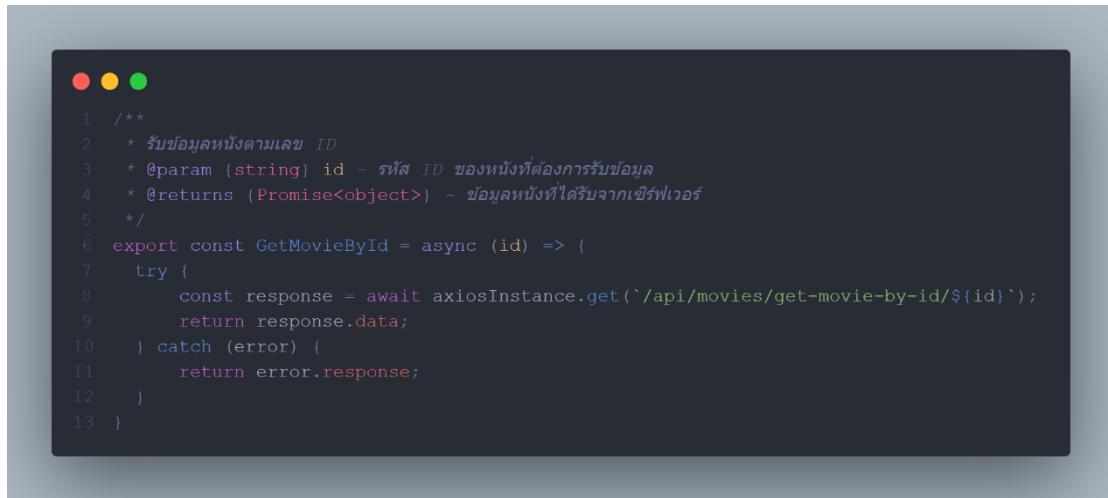
```

1  /**
2  * ลบรายการหนัง
3  * @param {object} payload - ข้อมูลเกี่ยวกับหนังที่ต้องการลบ
4  * @returns {Promise<object>} - ข้อมูลการลบหนังที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const DeleteMovie = async (payload) => {
7   try {
8     const response = await axiosInstance.post("/api/movies/delete-movie", payload);
9     return response.data;
10  } catch (error) {
11    return error.response;
12  }
13}

```

### movies.js (3/3)

5.GetMovieById: ฟังก์ชันนี้ใช้สำหรับรับข้อมูลของหนังโดยใช้เลข ID เมื่อเรียกใช้งาน ฟังก์ชันนี้จะทำการส่งคำขอ HTTP GET ไปยังเส้นทาง '/api/movies/get-movie-by-id/:id' บนเซิร์ฟเวอร์ โดย id คือรหัส ID ของหนังที่ต้องการรับข้อมูล และจะคืนข้อมูลของหนังที่ได้รับจากเซิร์ฟเวอร์กลับมา

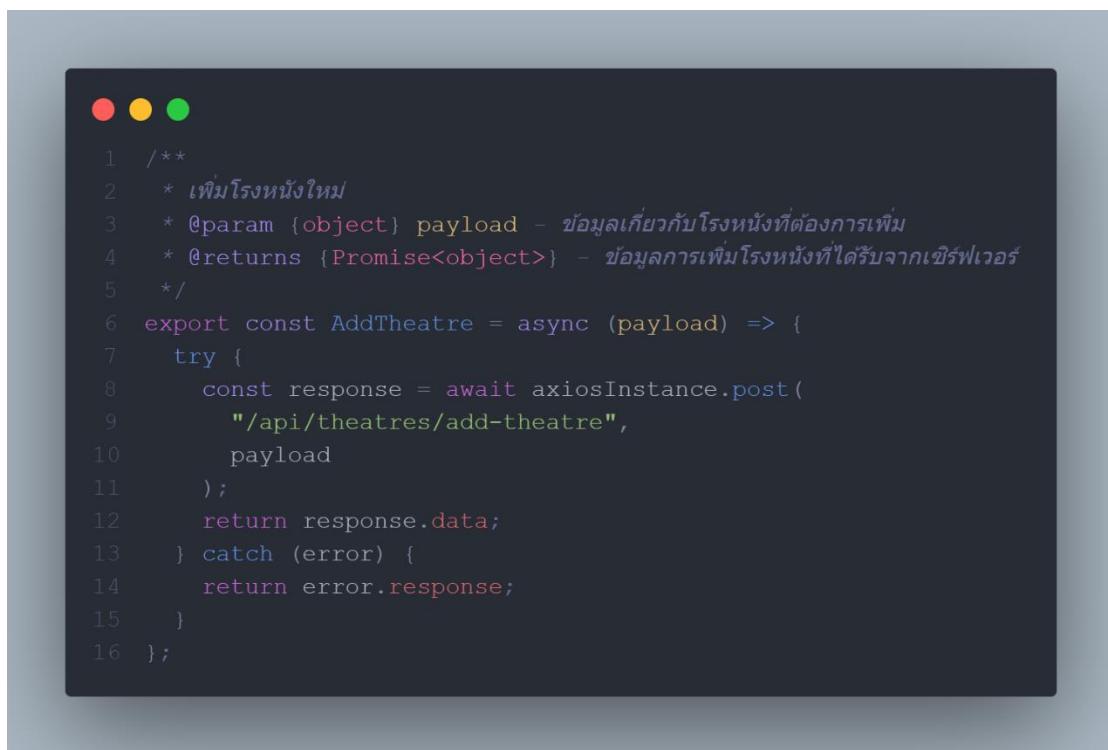


```
1  /**
2   * รับข้อมูลหนังตาม id
3   * @param {string} id - รหัส ID ของหนังที่ต้องการรับข้อมูล
4   * @returns {Promise<object>} - ข้อมูลหนังที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const GetMovieById = async (id) => {
7    try {
8      const response = await axiosInstance.get(`/api/movies/get-movie-by-id/${id}`);
9      return response.data;
10    } catch (error) {
11      return error.response;
12    }
13  }
```

## theatres.js (1/9)

คำอธิบาย : ไฟล์นี้เป็นโมดูล JavaScript ซึ่งมีหน้าที่เกี่ยวกับการจัดการกับ API ที่ให้บริการโดยใช้ Axios ในการส่งคำขอ HTTP ไปยังเซิร์ฟเวอร์ นอกจากนี้ยังมีฟังก์ชันต่างๆ ที่ใช้ในการเรียกใช้งาน API ต่างๆ โดยแต่ละฟังก์ชันจะมีหน้าที่และการทำงานดังนี้

1.AddTheatre: ใช้สำหรับเพิ่มข้อมูลโรงหนังใหม่เข้าสู่ระบบ โดยส่งคำขอ HTTP POST ไปยัง เซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/add-theatre' เพื่อเพิ่มโรงหนังใหม่ จากนั้นรับข้อมูลที่ได้รับจาก เซิร์ฟเวอร์และคืนค่ากลับ.



```

1  /**
2   * เพิ่มโรงหนังใหม่
3   * @param {object} payload - ข้อมูลเกี่ยวกับโรงหนังที่ต้องการเพิ่ม
4   * @returns {Promise<object>} - ข้อมูลการเพิ่มโรงหนังที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const AddTheatre = async (payload) => {
7    try {
8      const response = await axiosInstance.post(
9        "/api/theatres/add-theatre",
10       payload
11     );
12     return response.data;
13   } catch (error) {
14     return error.response;
15   }
16 };

```

## theatres.js (2/9)

2.GetAllTheatres: ใช้สำหรับแสดงรายการโรงหนังทั้งหมดที่มีในระบบ โดยส่งคำขอ HTTP GET ไปยังเชิร์ฟเวอร์ที่เส้นทาง '/api/theatres/get-all-theatres' จากนั้นรับข้อมูลรายการโรงหนังทั้งหมดที่ได้รับจากเชิร์ฟเวอร์และคืนค่ากลับ



```

1  /**
2   * แสดงรายการโรงหนังทั้งหมด
3   * @returns {Promise<object>} - ข้อมูลรายการโรงหนังทั้งหมดที่ได้รับจากเชิร์ฟเวอร์
4   */
5  export const GetAllTheatres = async () => {
6    try {
7      const response = await axiosInstance.get("/api/theatres/get-all-theatres");
8      return response.data;
9    } catch (error) {
10      return error.response;
11    }
12  };

```

3.GetAllTheatresByOwner: ใช้สำหรับแสดงรายการโรงหนังที่เป็นเจ้าของเท่านั้น โดยส่งคำขอ HTTP POST ไปยังเชิร์ฟเวอร์ที่เส้นทาง '/api/theatres/get-all-theatres-by-owner' จากนั้นรับข้อมูลรายการโรงหนังที่เป็นเจ้าของทั้งหมดที่ได้รับจากเชิร์ฟเวอร์และคืนค่ากลับ



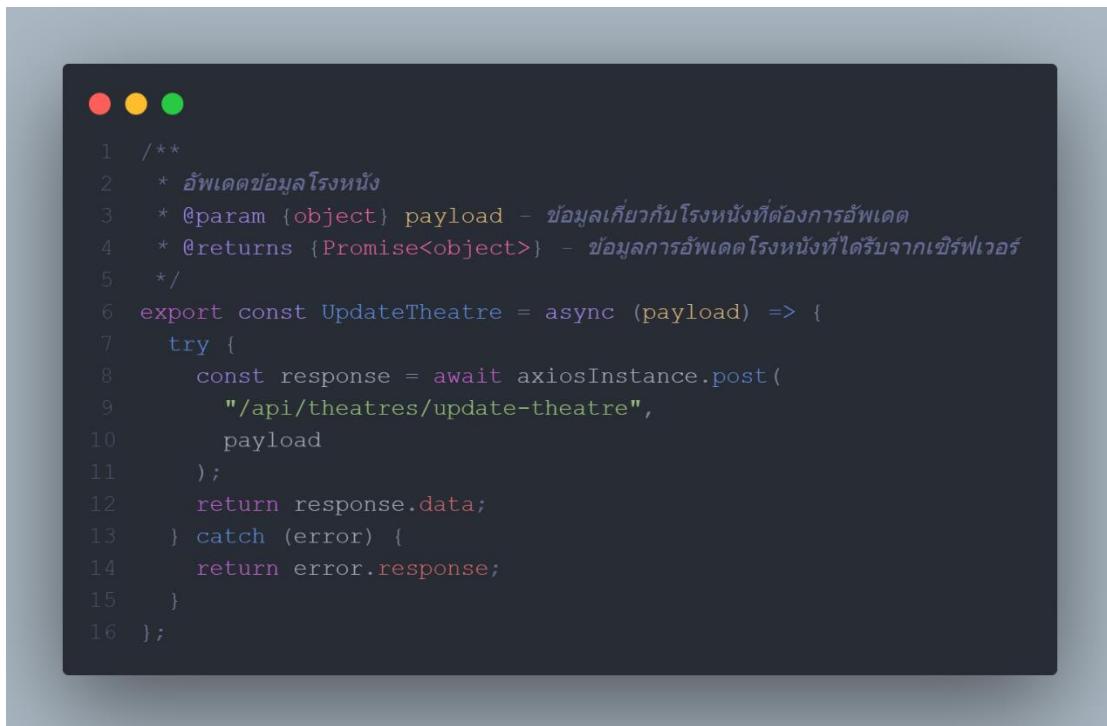
```

1  /**
2   * แสดงรายการโรงหนังที่ตัวเองเป็นเจ้าของเท่านั้น
3   * @param {object} payload - ข้อมูลเกี่ยวกับเจ้าของโรงหนัง
4   * @returns {Promise<object>} - ข้อมูลรายการโรงหนังที่เป็นเจ้าของที่ได้รับจากเชิร์ฟเวอร์
5   */
6  export const GetAllTheatresByOwner = async (payload) => {
7    try {
8      const response = await axiosInstance.post(
9        "/api/theatres/get-all-theatres-by-owner",
10        payload
11      );
12      return response.data;
13    } catch (error) {
14      return error.response;
15    }
16  };

```

### theatres.js (3/9)

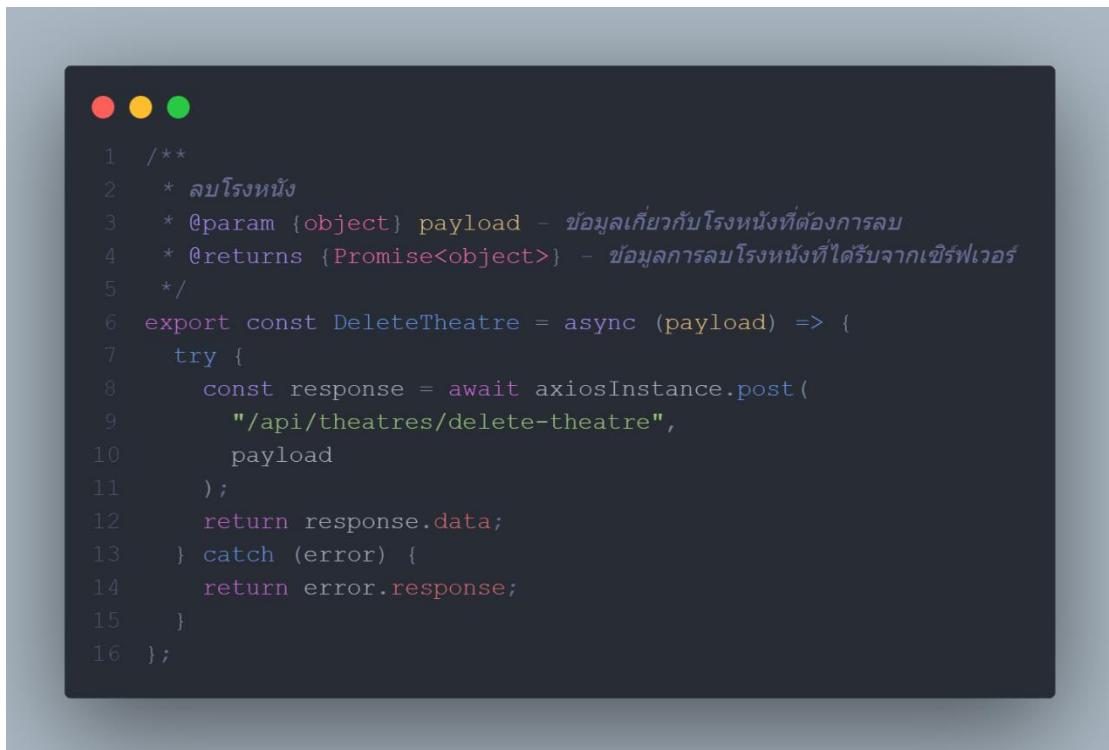
4. UpdateTheatre: ใช้สำหรับอัปเดตข้อมูลของโรงหนัง โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/update-theatre' จากนั้นรับข้อมูลการอัปเดตโรงหนังที่ได้รับจากเซิร์ฟเวอร์ และคืนค่ากลับ



```
1  /**
2   * อัปเดตข้อมูลโรงหนัง
3   * @param {object} payload - ข้อมูลเกี่ยวกับโรงหนังที่ต้องการอัปเดต
4   * @returns {Promise<object>} - ข้อมูลการอัปเดตโรงหนังที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const UpdateTheatre = async (payload) => {
7   try {
8     const response = await axiosInstance.post(
9       "/api/theatres/update-theatre",
10      payload
11    );
12     return response.data;
13   } catch (error) {
14     return error.response;
15   }
16};
```

### theatres.js (4/9)

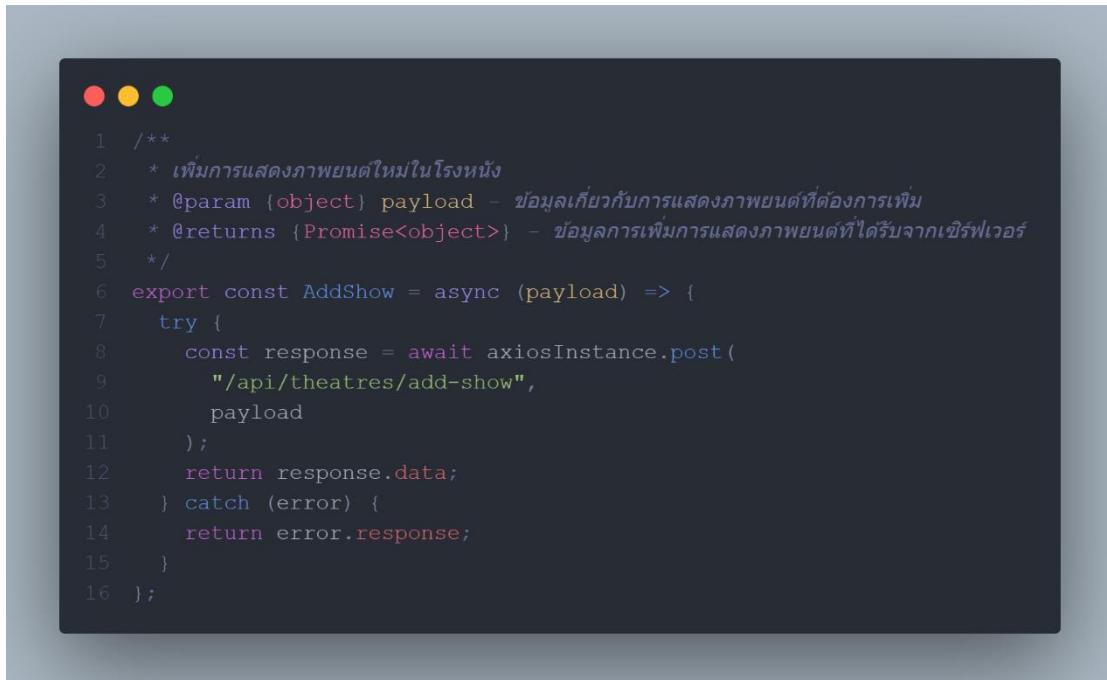
5. DeleteTheatre: ใช้สำหรับลบโรงหนังออกจากระบบ โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/delete-theatre' จากนั้นรับข้อมูลการลบโรงหนังที่ได้รับจากเซิร์ฟเวอร์และคืนค่ากลับ



```
1  /**
2   * ลบโรงหนัง
3   * @param {object} payload - ข้อมูลเกี่ยวกับโรงหนังที่ต้องการลบ
4   * @returns {Promise<object>} - ข้อมูลการลบโรงหนังที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const DeleteTheatre = async (payload) => {
7    try {
8      const response = await axiosInstance.post(
9        "/api/theatres/delete-theatre",
10       payload
11     );
12     return response.data;
13   } catch (error) {
14     return error.response;
15   }
16 };
```

### theatres.js (5/9)

6. AddShow: ใช้สำหรับเพิ่มการแสดงภาพยนตร์ใหม่ในโรงหนัง โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/add-show' จากนั้นรับข้อมูลการเพิ่มการแสดงภาพยนตร์ที่ได้รับจากเซิร์ฟเวอร์และคืนค่ากลับ



```
1  /**
2   * เพิ่มการแสดงภาพยนตร์ใหม่ในโรงหนัง
3   * @param {object} payload - ข้อมูลเกี่ยวกับการแสดงภาพยนตร์ที่ต้องการเพิ่ม
4   * @returns {Promise<object>} - ข้อมูลการเพิ่มการแสดงภาพยนตร์ที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const AddShow = async (payload) => {
7    try {
8      const response = await axiosInstance.post(
9        "/api/theatres/add-show",
10       payload
11     );
12     return response.data;
13   } catch (error) {
14     return error.response;
15   }
16};
```

## theatres.js (6/9)

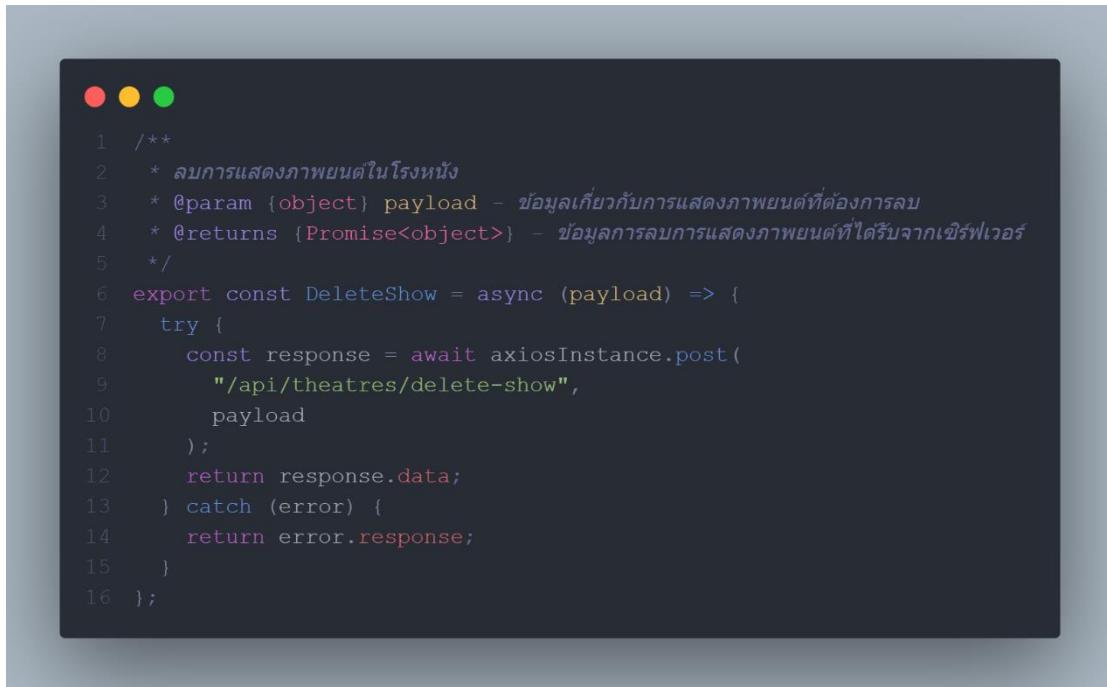
7. GetAllShowsByTheatre: ใช้สำหรับแสดงรายการภาพยนตร์ที่มีอยู่ในโรงหนัง โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/get-all-shows-by-theatre' จากนั้นรับข้อมูลรายการภาพยนตร์ที่มีอยู่ในโรงหนังที่ได้รับจากเซิร์ฟเวอร์และคืนค่ากลับ



```
1  /**
2   * แสดงรายการภาพยนตร์ที่มีอยู่ในโรงหนัง
3   * @param {object} payload - ข้อมูลเกี่ยวกับโรงหนังที่ต้องการดูรายการภาพยนตร์
4   * @returns {Promise<object>} - ข้อมูลรายการภาพยนตร์ที่มีอยู่ในโรงหนังที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const GetAllShowsByTheatre = async (payload) => {
7   try {
8     const response = await axiosInstance.post(
9       "/api/theatres/get-all-shows-by-theatre",
10      payload
11    );
12    return response.data;
13  } catch (error) {
14    return error.response;
15  }
16};
```

## theatres.js (7/9)

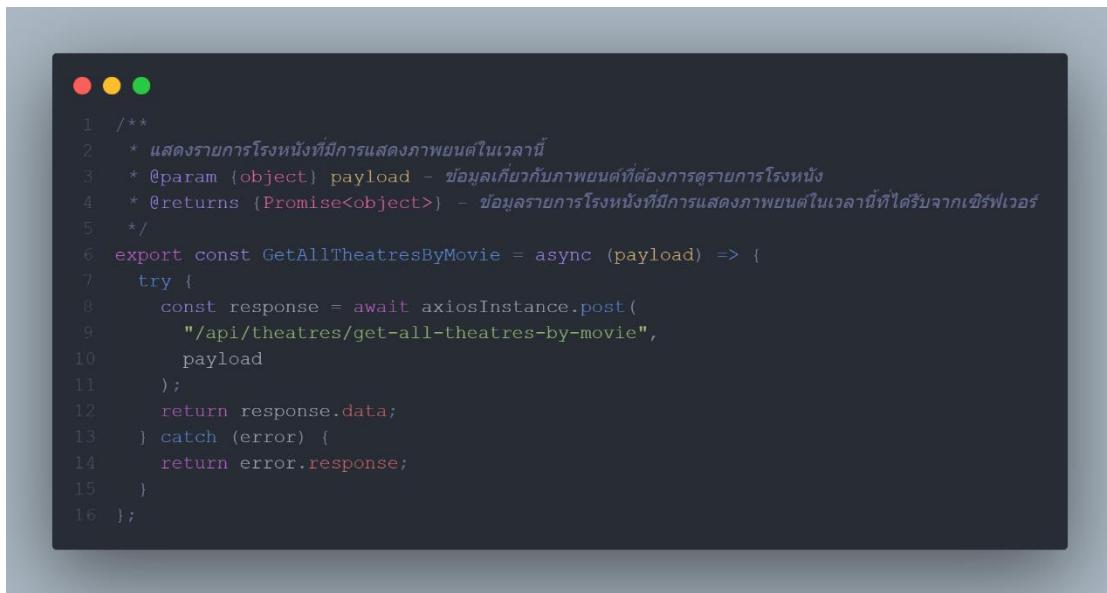
8. DeleteShow: ใช้สำหรับลบการแสดงภาพยนตร์ในโรงหนัง โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/delete-show' จากนั้นรับข้อมูลการลบการแสดงภาพยนตร์ที่ได้รับจากเซิร์ฟเวอร์และคืนค่ากลับ



```
1  /**
2   * ลบการแสดงภาพยนตร์ในโรงหนัง
3   * @param {object} payload - ข้อมูลเกี่ยวกับการแสดงภาพยนตร์ที่ต้องการลบ
4   * @returns {Promise<object>} - ข้อมูลการลบการแสดงภาพยนตร์ที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const DeleteShow = async (payload) => {
7    try {
8      const response = await axiosInstance.post(
9        "/api/theatres/delete-show",
10       payload
11     );
12     return response.data;
13   } catch (error) {
14     return error.response;
15   }
16};
```

## theatres.js (8/9)

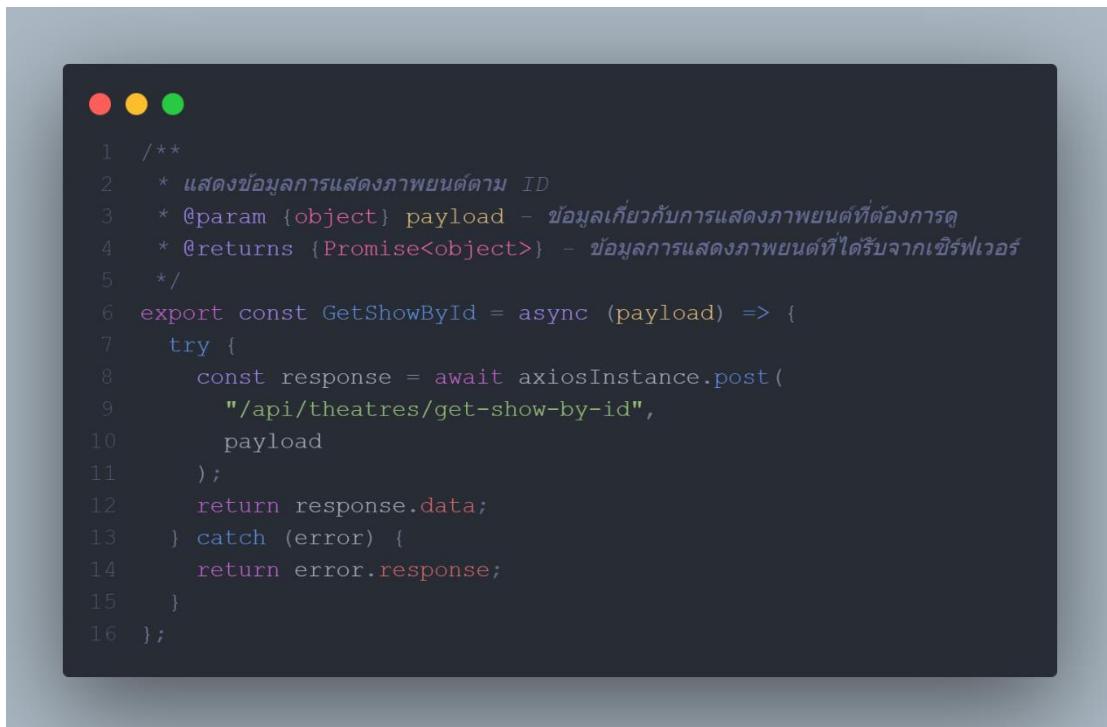
9. GetAllTheatresByMovie: ใช้สำหรับแสดงรายการโรงหนังที่มีการแสดงภาพยนตร์ในเวลานี้ โดยส่งคำขอ HTTP POST 'ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/get-all-theatres-by-movie' จากนั้นรับข้อมูลรายการโรงหนังที่มีการแสดงภาพยนตร์ในเวลานี้ที่ได้รับจากเซิร์ฟเวอร์และคืนค่ากลับ



```
1  /**
2   * แสดงรายการโรงหนังที่มีการแสดงภาพยนตร์ในเวลานี้
3   * @param {object} payload - ข้อมูลเกี่ยวกับภาพยนตร์ที่ต้องการดูรายการโรงหนัง
4   * @returns {Promise<object>} - ข้อมูลรายการโรงหนังที่มีการแสดงภาพยนตร์ในเวลานี้ที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const GetAllTheatresByMovie = async (payload) => {
7   try {
8     const response = await axiosInstance.post(
9       "/api/theatres/get-all-theatres-by-movie",
10      payload
11    );
12    return response.data;
13  } catch (error) {
14    return error.response;
15  }
16};
```

## theatres.js (9/9)

10. GetShowById: ใช้สำหรับแสดงข้อมูลการแสดงภาพยนตร์ตาม ID โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/theatres/get-show-by-id' จากนั้นรับข้อมูลการแสดงภาพยนตร์ที่ได้รับจากเซิร์ฟเวอร์และคืนค่ากลับ



```
1  /**
2   * แสดงข้อมูลการแสดงภาพยนตร์ตาม ID
3   * @param {object} payload - ข้อมูลเกี่ยวกับการแสดงภาพยนตร์ที่ต้องการดู
4   * @returns {Promise<object>} - ข้อมูลการแสดงภาพยนตร์ที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const GetShowById = async (payload) => {
7    try {
8      const response = await axiosInstance.post(
9        "/api/theatres/get-show-by-id",
10       payload
11     );
12     return response.data;
13   } catch (error) {
14     return error.response;
15   }
16};
```

## users.js (1/2)

**คำอธิบาย :** ไฟล์นี้มีฟังก์ชันที่เกี่ยวกับการจัดการผู้ใช้ในระบบ โดยทุกฟังก์ชันจะส่งคำขอ HTTP ไปยังเซิร์ฟเวอร์ที่ระบุและจะรับข้อมูลผู้ใช้หรือข้อผิดพลาดที่เกิดขึ้นกลับมา เช่น การลงทะเบียนผู้ใช้ใหม่ (RegisterUser), เข้าสู่ระบบ (LoginUser), และการรับข้อมูลผู้ใช้ปัจจุบัน (GetCurrentUser) ดังนี้

1. RegisterUser: ใช้สำหรับลงทะเบียนผู้ใช้ใหม่ในระบบ โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่เส้นทาง '/api/users/register' เพื่อส่งข้อมูลการลงทะเบียน ถ้าลงทะเบียนสำเร็จ จะคืนข้อมูลผู้ใช้ที่ลงทะเบียนเป็น JSON แต่หากเกิดข้อผิดพลาดขณะลงทะเบียน จะส่งคืนข้อมูลผิดพลาดจากเซิร์ฟเวอร์



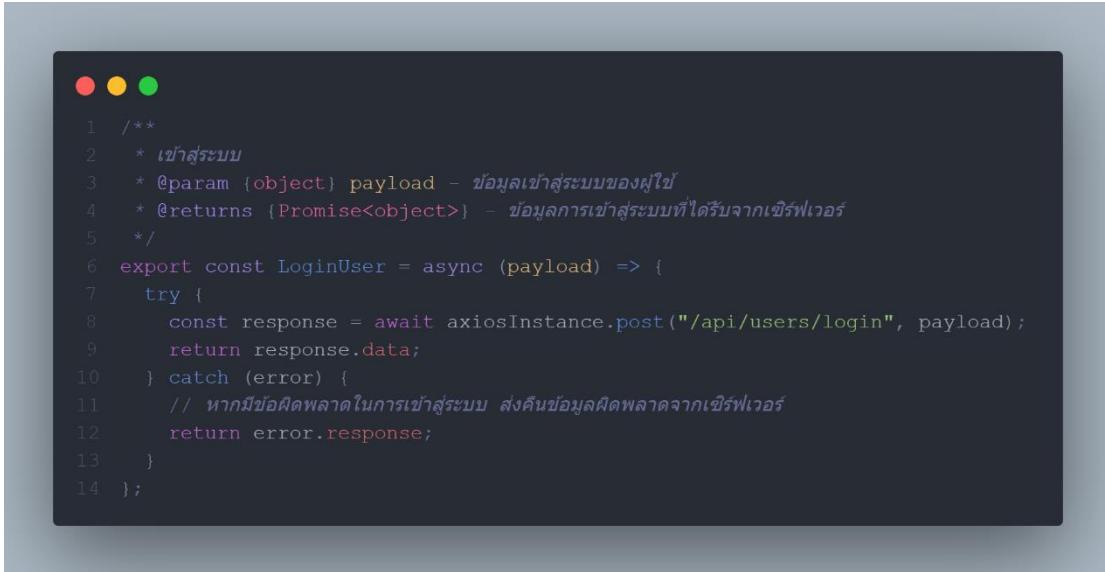
```

1  /**
2  * ลงทะเบียนผู้ใช้ใหม่
3  * @param {object} payload - ข้อมูลเกี่ยวกับผู้ใช้ที่ต้องการลงทะเบียน
4  * @returns {Promise<object>} - ข้อมูลการลงทะเบียนผู้ใช้ที่ได้รับจากเซิร์ฟเวอร์
5  */
6 export const RegisterUser = async (payload) => {
7   try {
8     const response = await axiosInstance.post("/api/users/register", payload);
9     return response.data;
10  } catch (error) {
11    // หากมีข้อผิดพลาดในการลงทะเบียน ส่งคืนข้อมูลผิดพลาดจากเซิร์ฟเวอร์
12    return error.response;
13  }
14 };

```

## users.js (2/2)

2. LoginUser: ใช้สำหรับเข้าสู่ระบบผู้ใช้ โดยส่งคำขอ HTTP POST ไปยังเซิร์ฟเวอร์ที่สั่นทาง '/api/users/login' เพื่อส่งข้อมูลการเข้าสู่ระบบ ถ้าเข้าสู่ระบบสำเร็จ จะคืนข้อมูลการเข้าสู่ระบบเป็น JSON แต่หากเกิดข้อผิดพลาดจะ返ข้อมูลการเข้าสู่ระบบ จะส่งคืนข้อมูลผิดพลาดจากเซิร์ฟเวอร์



```

1  /**
2   * เข้าสู่ระบบ
3   * @param {object} payload - ข้อมูลเข้าสู่ระบบของผู้ใช้
4   * @returns {Promise<object>} - ข้อมูลการเข้าสู่ระบบที่ได้รับจากเซิร์ฟเวอร์
5   */
6  export const LoginUser = async (payload) => {
7    try {
8      const response = await axiosInstance.post("/api/users/login", payload);
9      return response.data;
10    } catch (error) {
11      // หากมีข้อผิดพลาดในการเข้าสู่ระบบ ส่งคืนข้อมูลผิดพลาดจากเซิร์ฟเวอร์
12      return error.response;
13    }
14  };

```

3. GetCurrentUser: ใช้สำหรับรับข้อมูลผู้ใช้ปัจจุบันที่เข้าสู่ระบบ โดยส่งคำขอ HTTP GET ไปยังเซิร์ฟเวอร์ที่สั่นทาง '/api/users/get-current-user' ถ้าข้อมูลผู้ใช้ถูกต้อง จะคืนข้อมูลผู้ใช้ปัจจุบันเป็น JSON แต่หากเกิดข้อผิดพลาดจะ返ข้อมูลผู้ใช้ จะส่งคืนข้อมูลผิดพลาด



```

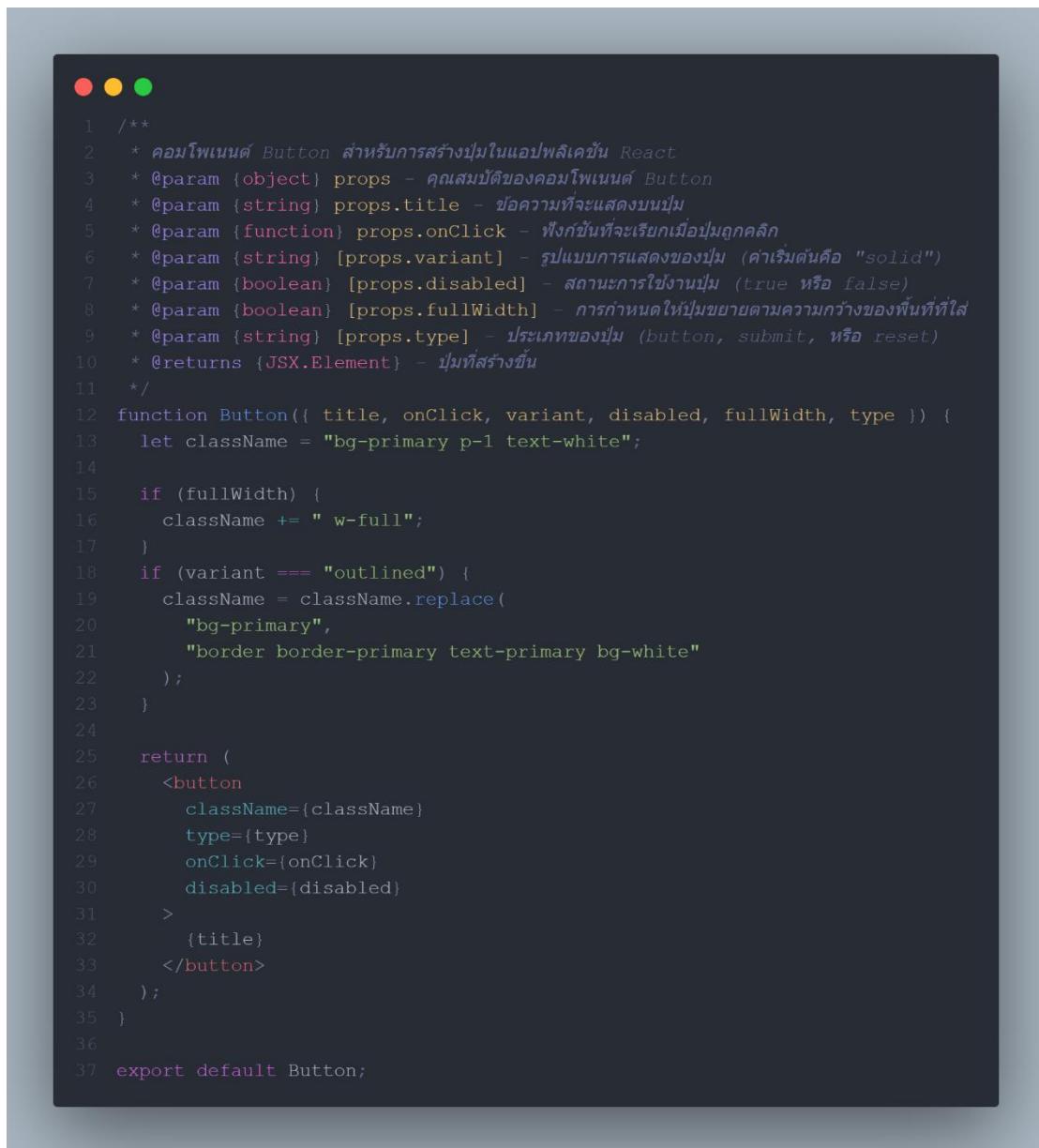
1  /**
2   * รับข้อมูลผู้ใช้ปัจจุบัน
3   * @returns {Promise<object>} - ข้อมูลผู้ใช้ปัจจุบันที่ได้รับจากเซิร์ฟเวอร์
4   */
5  export const GetCurrentUser = async () => {
6    try {
7      const response = await axiosInstance.get("/api/users/get-current-user");
8      return response.data;
9    } catch (error) {
10      // หากมีข้อผิดพลาดในการรับข้อมูลผู้ใช้ปัจจุบัน ส่งคืนข้อมูลผิดพลาด
11      return error;
12    }
13  };

```

## ส่วนประกอบ (Components)

### Button.js

คำอธิบาย : ไฟล์นี้เป็นโมดูล React ที่มีประโยชน์ในการสร้างปุ่มในแอปพลิเคชัน React โดยจะสร้างปุ่มตามค่าที่ได้รับผ่าน props ที่ระบุไว้ในฟังก์ชัน Button โดย Button({ title, onClick, variant, disabled, fullWidth, type }) เป็นฟังก์ชันหลักที่ใช้สร้างคอมโพเนนต์



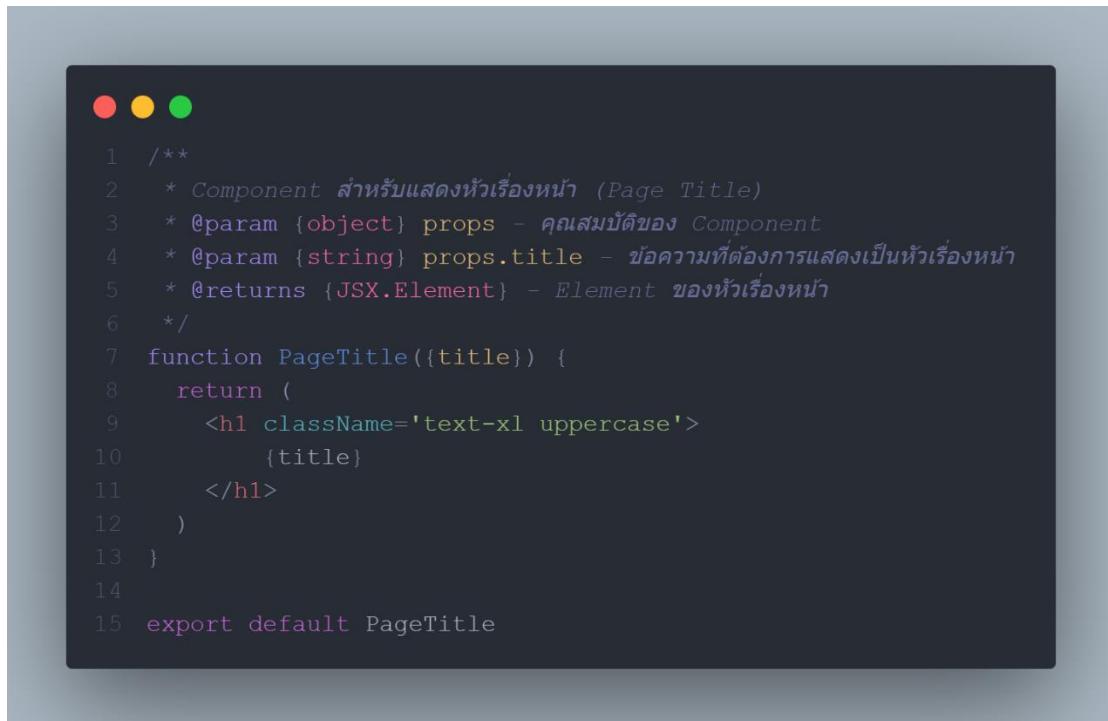
```

1  /**
2   * คอมโพเนนต์ Button สำหรับการสร้างปุ่มในแอปพลิเคชัน React
3   * @param {object} props - คุณสมบัติของคอมโพเนนต์ Button
4   * @param {string} props.title - ข้อความที่จะแสดงบนปุ่ม
5   * @param {function} props.onClick - ฟังก์ชันที่จะเรียกเมื่อปุ่มถูกคลิก
6   * @param {string} [props.variant] - รูปแบบการแสดงของปุ่ม (ค่าเริ่มต้นคือ "solid")
7   * @param {boolean} [props.disabled] - สภาวะการใช้งานปุ่ม (true หรือ false)
8   * @param {boolean} [props.fullWidth] - การกำหนดให้ปุ่มขยายตามความกว้างของพื้นที่ที่ใส่
9   * @param {string} [props.type] - ประเภทของปุ่ม (button, submit, หรือ reset)
10  * @returns {JSX.Element} - ปุ่มที่สร้างขึ้น
11 */
12 function Button({ title, onClick, variant, disabled, fullWidth, type }) {
13   let className = "bg-primary p-1 text-white";
14
15   if (fullWidth) {
16     className += " w-full";
17   }
18   if (variant === "outlined") {
19     className = className.replace(
20       "bg-primary",
21       "border border-primary text-primary bg-white"
22     );
23   }
24
25   return (
26     <button
27       className={className}
28       type={type}
29       onClick={onClick}
30       disabled={disabled}
31     >
32       {title}
33     </button>
34   );
35 }
36
37 export default Button;

```

## PageTitle.js

คำอธิบาย : ไฟล์นี้เป็นคอมโพเนนต์ React ที่มีชื่อว่า PageTitle ซึ่งใช้สำหรับแสดงหัวเรื่องหน้า (Page Title) ของแอปพลิเคชัน React โดย title คือข้อความที่ต้องการแสดงเป็นหัวเรื่องหน้า ซึ่งใช้ className เพื่อกำหนดลักษณะของข้อความ ซึ่งในที่นี้ใช้ text-xl เพื่อกำหนดขนาดของข้อความให้ใหญ่ขึ้นและ uppercase เพื่อกำหนดให้ข้อความแสดงเป็นตัวพิมพ์ใหญ่ทั้งหมด



```
1  /**
2   * Component สำหรับแสดงหัวเรื่องหน้า (Page Title)
3   * @param {object} props - คุณสมบัติของ Component
4   * @param {string} props.title - ข้อความที่ต้องการแสดงเป็นหัวเรื่องหน้า
5   * @returns {JSX.Element} - Element ของหัวเรื่องหน้า
6  */
7  function PageTitle({title}) {
8    return (
9      <h1 className='text-xl uppercase'>
10        {title}
11      </h1>
12    )
13  }
14
15 export default PageTitle
```

## ProtectedRoute.js (1/2)

**คำอธิบาย :** ไฟล์นี้เป็นส่วนหนึ่งของ React application ซึ่งมีหน้าที่จัดการเส้นทางที่มีการควบคุม การเข้าถึงหน้าจอในสถานะที่ต้องมีการเข้าสู่ระบบ โดยมีคุณสมบัติ (Props) children เป็น element ภายในของ Component ซึ่งจะถูก render เมื่อสถานะการเข้าสู่ระบบถูกตรวจสอบแล้ว และฟังก์ชัน getCurrentUser ใช้สำหรับเรียกข้อมูลผู้ใช้ปัจจุบันจากเซิร์ฟเวอร์ โดยใช้ฟังก์ชัน GetUser ที่ได้นำเข้ามาจากการ调用 apicalls/users ถ้าเรียกข้อมูลผู้ใช้สำเร็จ จะเก็บข้อมูลผู้ใช้ใน Redux state และแสดงผลตามสิทธิ์ของผู้ใช้ แต่หากเรียกข้อมูลผู้ใช้ไม่สำเร็จ จะแสดงข้อความผิดพลาดและทำการลบ token ใน localStorage และเปลี่ยนเส้นทางไปยังหน้า login



```

1  /**
2   * Component สำหรับการจัดการเส้นทางที่มีการควบคุมการเข้าถึงหน้าจอในสถานะที่ต้องมีการเข้าสู่ระบบ
3   * @param {object} props - คุณสมบัติของ Component
4   * @param {JSX.Element} props.children - Element ภายในของ Component
5   * @returns {JSX.Element} - Element ที่มีการควบคุมการเข้าถึงหน้าจอในสถานะที่ต้องมีการเข้าสู่ระบบ
6  */
7  function ProtectedRoute({ children }) {
8    const { user } = useSelector((state) => state.users);
9    const navigate = useNavigate();
10   const dispatch = useDispatch();
11
12  /**
13   * ฟังก์ชันสำหรับการเรียกข้อมูลผู้ใช้ปัจจุบันจากเซิร์ฟเวอร์
14   */
15  const getCurrentUser = async () => {
16    try {
17      dispatch>ShowLoading();
18      const response = await GetUser();
19      dispatch(HideLoading());
20      if (response.success) {
21        dispatch(SetUser(response.data));
22      } else {
23        dispatch(SetUser(null));
24        message.error(response.message);
25        localStorage.removeItem("token");
26        navigate("/login");
27      }
28    } catch (error) {
29      dispatch(SetUser(null));
30      dispatch(HideLoading());
31      message.error(error.message);
32    }
33  };

```

### ProtectedRoute.js (2/3)

useEffect ใช้ในการตรวจสอบว่าผู้ใช้เข้าสู่ระบบหรือไม่ โดยตรวจสอบว่ามี token ใน localStorage หรือไม่ ถ้ามี token จะเรียกใช้งานฟังก์ชัน getCurrentUser เพื่อเช็คสถานะของผู้ใช้ หากไม่มี token จะเปลี่ยนเส้นทางไปยังหน้า login



```

1 // ทำการตรวจสอบว่าผู้ใช้เข้าสู่ระบบหรือยัง โดยใช้ localStorage.getItem("token")
2 useEffect(() => {
3   if (localStorage.getItem("token")) {
4     getCurrentUser();
5   } else {
6     navigate("/login");
7   }
8 }, []);

```

หากมีข้อมูลผู้ใช้ใน Redux state จะแสดง header ด้านบนของหน้าเพื่อแสดงข้อมูลผู้ใช้และเมนูการนำทาง เมนูการนำทางจะแสดงตามสิทธิ์ของผู้ใช้ (admin, พนักงาน, หรือผู้ใช้ทั่วไป) หากคลิกที่ชื่อผู้ใช้ จะเปลี่ยนเส้นทางไปยังหน้าที่เหมาะสมตามสิทธิ์ของผู้ใช้ แต่ถ้าคลิกที่ไอคอน logout จะทำการลบ token ใน localStorage และเปลี่ยนเส้นทางไปยังหน้า login

```

1   return (
2     user && (
3       <div className="layout p-1">
4         <div className="header bg-primary flex justify-between p-2">
5           <div>
6             <h1
7               className="text-2xl text-white cursor-pointer"
8               onClick={() => navigate("/")}>
9             >
10            { " " }
11            HOMEPAGE{ " " }
12          </h1>
13        </div>
14
15        <div className="bg-white p-1 flex gap-1">
16          <i className="ri-shield-user-line text-primary"></i>
17          <h1
18            className="text-sm underline"
19            onClick={() => {
20              if (user.isAdmin) {
21                // หากผู้ใช้เป็นผู้ดูแล  จะเปลี่ยนเส้นทางไปยังหน้า admin
22                navigate("/admin");
23              } else if (user.isEmployee) {
24                // หากผู้ใช้เป็นพนักงาน จะเปลี่ยนเส้นทางไปยังหน้า theatres
25                navigate("/theatres");
26              }
27            else {
28              // หากผู้ใช้เป็นผู้ใช้ทั่วไป จะเปลี่ยนเส้นทางไปยังหน้า profile
29              navigate("/profile");
30            }
31          } }
32        >
33          {user.name}
34        </h1>
35
36        <i
37          className="ri-logout-box-r-line ml-2"
38          onClick={() => {
39            localStorage.removeItem("token");
40            navigate("/login");
41          }}
42        ></i>
43      </div>
44    </div>
45    <div className="content mt-1 p-1">{children}</div>
46  </div>
47 )
48 );
49 }
50
51 export default ProtectedRoute;

```

## ส่วนผู้ดูแลระบบ (Admin Section)

### index.js

**คำอธิบาย :** ไฟล์นี้เป็นส่วนหนึ่งของโปรแกรม React ซึ่งใช้สร้างหน้า Admin Control Panel โดยมีการนำเข้า component ต่าง ๆ เช่น Tabs, PageTitle, MoviesList, และ TheatresList เพื่อให้ผู้ดูแลระบบสามารถดูแลและจัดการรายการหนังและโรงหนังได้อย่างสะดวก โดยแสดงหัวเรื่องหน้า Admin Control Panel และแท็บสำหรับเลือกแสดงรายการหนังหรือโรงหนัง ซึ่งแต่ละแท็บจะแสดงรายการตามที่ผู้ใช้เลือกโดยใช้ MoviesList และ TheatresList ตามลำดับในแต่ละแท็บจะ



```

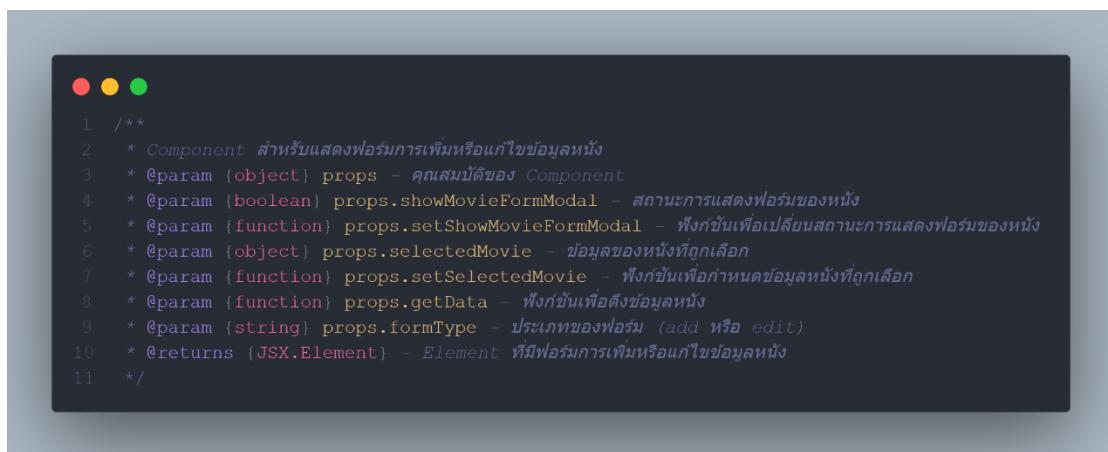
1  /**
2   * Component สำหรับหน้า Admin Control Panel ที่มีการแสดงรายการหนังและโรงหนัง
3   * @returns {JSX.Element} - Element ที่มีหน้า Admin Control Panel
4   * ที่ประกอบด้วยหัวเรื่องหน้า และแท็บสำหรับแสดงรายการหนังและโรงหนัง
5   */
6  function Admin() {
7    return (
8      <div>
9        {/* แสดงหัวเรื่องหน้า Admin Control Panel */}
10       <PageTitle title="Admin Control Panel" />
11
12       {/* แสดงแท็บสำหรับเลือกแสดงรายการหนังหรือโรงหนัง */}
13       <Tabs defaultActiveKey="1">
14
15         {/* แสดงแท็บสำหรับรายการหนัง */}
16         <Tabs.TabPane tab="Movies" key="1">
17           <MoviesList />
18         </Tabs.TabPane>
19
20         {/* แสดงแท็บสำหรับรายการโรงหนัง */}
21         <Tabs.TabPane tab="Theatres" key="2">
22           <TheatresList />
23         </Tabs.TabPane>
24
25       </Tabs>
26     </div>
27   );
28 }
29
30 export default Admin;

```

## MovieForm.js

**คำอธิบาย :** ไฟล์นี้เป็น Component สำหรับแสดงฟอร์มการเพิ่มหรือแก้ไขข้อมูลหนัง

โดยมีการใช้งาน React Hooks เช่น useState เพื่อจัดการสถานะของฟอร์มและข้อมูล และใช้งาน useEffect เพื่อดำเนินการหลังจากที่ Component ถูก Render โดยรวมแล้ว Component นี้มีลักษณะการทำงานในรูปแบบต่อเนื่อง โดยเริ่มต้นจากการตรวจสอบข้อมูลหนังที่ถูกเลือกและจัดรูปแบบข้อมูลวันที่ จากนั้น ทำการส่งข้อมูลฟอร์มไปยัง API เพื่อเพิ่มหรือแก้ไขข้อมูล และรอรับคำตอบจาก API เพื่อดำเนินการต่อตามผลลัพธ์ที่ได้รับ สุดท้าย Component จะแสดงฟอร์มการเพิ่มหรือแก้ไขข้อมูลหนังในรูปแบบ Modal พร้อมกับปุ่มสำหรับยืนยันหรือยกเลิกการกรอกข้อมูล



```

1  /**
2   * Component สำหรับแสดงฟอร์มการเพิ่มหรือแก้ไขข้อมูลหนัง
3   * @param {object} props - คุณสมบัติของ Component
4   * @param {boolean} props.showMovieFormModal - สถานะการแสดงฟอร์มนี้
5   * @param {function} props.setShowMovieFormModal - ฟังก์ชันเพื่อเปลี่ยนสถานะการแสดงฟอร์มนี้
6   * @param {object} props.selectedMovie - ข้อมูลของหนังที่ถูกเลือก
7   * @param {function} props.setSelectedMovie - ฟังก์ชันเพื่อเปลี่ยนหนังที่ถูกเลือก
8   * @param {function} props.getData - ฟังก์ชันเพื่อดึงข้อมูลหนังที่ถูกเลือก
9   * @param {string} props.formType - ประเภทของฟอร์ม (add หรือ edit)
10  * @returns {JSX.Element} - Element ที่มีฟอร์มการเพิ่มหรือแก้ไขข้อมูลหนัง
11 */

```

```

1  function MovieForm({
2    showMovieFormModal,
3    setShowMovieFormModal,
4    selectedMovie,
5    setSelectedMovie,
6    getData,
7    formType,
8  }) {
9    // ตรวจสอบและจัดรูปแบบวันที่ releaseDate หากมี selectedMovie
10   if (selectedMovie) {
11     selectedMovie.releaseDate = moment(selectedMovie.releaseDate).format(
12       "YYYY-MM-DD"
13     );
14   }
15
16   const dispatch = useDispatch();
17
18   /**
19    * ฟังก์ชันสำหรับการส่งข้อมูลฟอร์มการเพิ่มหรือแก้ไขข้อมูลหนัง
20    * @param {object} values - ข้อมูลที่ถูกส่งมาจากฟอร์ม
21    */
22   const onFinish = async (values) => {
23     try {
24       dispatch>ShowLoading(); // แสดง Loading
25       let response = null;
26
27       // ตรวจสอบ formType เพื่อดำเนินการเพิ่มหรือแก้ไขข้อมูล
28       if (formType === "add") {
29         response = await AddMovie(values);
30       } else {
31         response = await UpdateMovie({
32           ...values,
33           movieId: selectedMovie._id,
34         });
35       }
36
37       // ตรวจสอบค่าตอบจาก API และดำเนินการต่อ
38       if (response.success) {
39         getData();
40         message.success(response.message);
41         setShowMovieFormModal(false);
42       } else {
43         message.error(response.message);
44       }
45       dispatch(HideLoading()); // ปิด Loading
46     } catch (error) {
47       // ปิด Loading และแสดงข้อความ error
48       dispatch(HideLoading());
49       message.error(error.message);
50     }
51   };

```

```

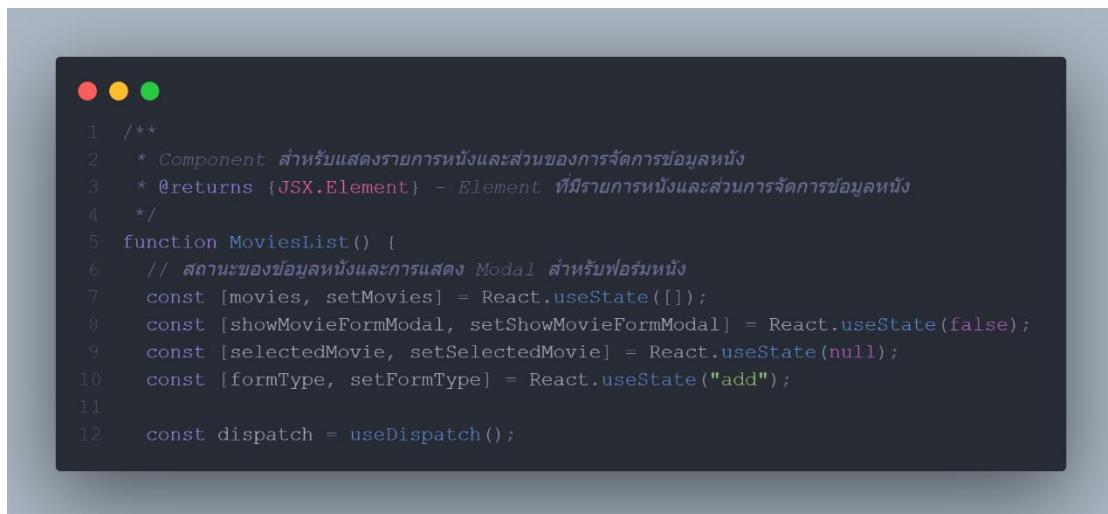
1  return (
2    <Modal
3      title={formType === "add" ? "ADD MOVIE" : "EDIT MOVIE"}
4      open={showMovieFormModal}
5      onCancel={() => {
6        // ປຶກ Modal ແລະ ຂໍສິນ selectedMovie ມີນ null
7        setShowMovieFormModal(false);
8        setSelectedMovie(null);
9      }}
10     footer={null}
11     width={800}
12   >
13   <Form layout="vertical" onFinish={onFinish} initialValues={selectedMovie}>
14     <Row gutter={16}>
15       <Col span={24}>
16         <Form.Item label="Movie Name" name="title">
17           <input type="text" />
18         </Form.Item>
19       </Col>
20
21       <Col span={24}>
22         <Form.Item label="Movie Description" name="description">
23           <textarea type="text" />
24         </Form.Item>
25       </Col>
26
27       <Col span={8}>
28         <Form.Item label="Movie Duration (mins)" name="duration">
29           <input type="number" />
30         </Form.Item>
31       </Col>
32
33     {/* ແສດງຕົວເລືອກຂອງແຕລະ Genre */}
34     <Col span={8}>
35       <Form.Item label="Genre" name="genre">
36         <select name="" id="">
37           <option value="">Select Genre</option>
38           <option value="Action">Action</option>
39           <option value="Adventure">Adventure</option>
40           <option value="Animation">Animation</option>
41           <option value="Comedy">Comedy</option>
42           <option value="Documentary">Documentary</option>
43           <option value="Drama">Drama</option>
44           <option value="Experimental">Experimental</option>
45           <option value="Fantasy">Fantasy</option>
46           <option value="Historical">Historical</option>
47           <option value="Horror">Horror</option>
48           <option value="Musical">Musical</option>
49           <option value="Mystery">Mystery</option>
50           <option value="Parody">Parody</option>
51           <option value="Romance">Romance</option>
52           <option value="Sports">Sports</option>
53           <option value="Thriller">Thriller</option>
54           <option value="Western">Western</option>
55         </select>
56       </Form.Item>
57     </Col>

```

```
1      /* ແສດງຕົວເລືອກຂອງແຕ່ລະ Language */
2      <Col span={8}>
3          <Form.Item label="Language" name="language">
4              <select name="" id="">
5                  <option value="">Select Language</option>
6                  <option value="Thai">Thai</option>
7                  <option value="English">English</option>
8                  <option value="Japan">Japan</option>
9              </select>
10         </Form.Item>
11     </Col>
12
13     <Col span={8}>
14         <Form.Item label="Movie Release Date" name="releaseDate">
15             <input type="date" />
16         </Form.Item>
17     </Col>
18
19     <Col span={16}>
20         <Form.Item label="Poster URL" name="poster">
21             <input type="text" />
22         </Form.Item>
23     </Col>
24 </Row>
25
26 <div className="flex justify-end gap-1">
27     /* ຜົນ Cancel ເພື່ອຍັກເລີກກາຮຽກອົບນ້ອມລູກ */
28     <Button
29         title="Cancel"
30         variant="outlined"
31         type="button"
32         onClick={() => {
33             setShowMovieFormModal(false);
34             setSelectedMovie(null);
35         }}
36     />
37     /* ຜົນ Save ເພື່ອຢືນບັນກາຮຽກອົບນ້ອມລູກ */
38     <Button title="Save" type="submit" />
39 </div>
40     </Form>
41 </Modal>
42 );
43 }
44
45 export default MovieForm;
```

## MoviesList.js

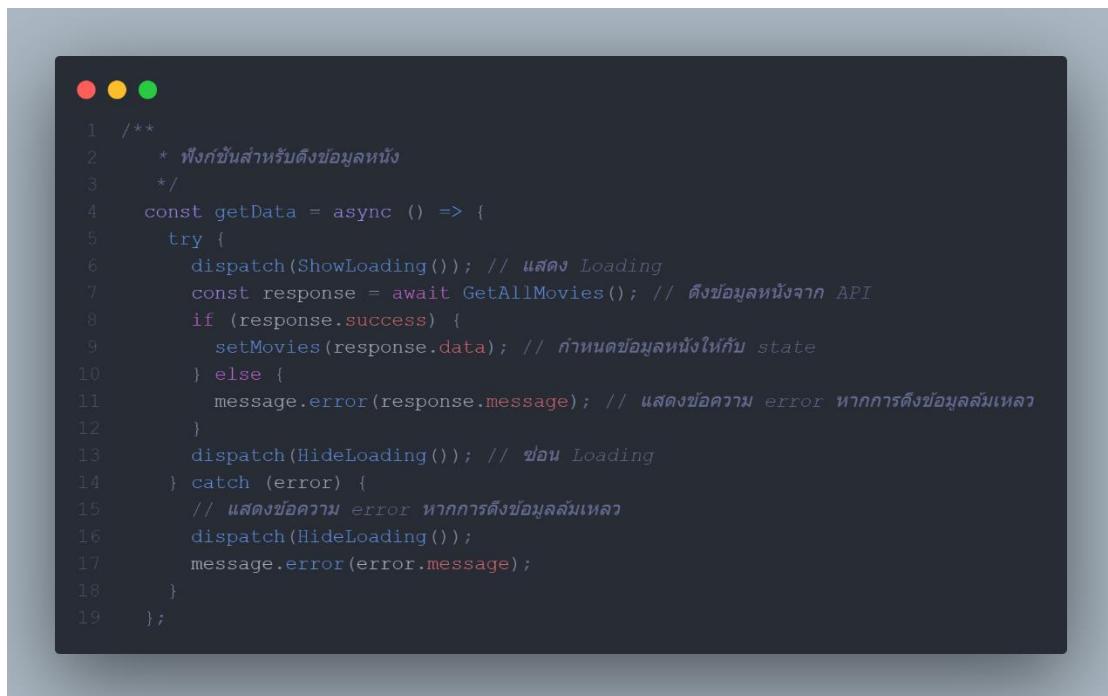
**คำอธิบาย :** ไฟล์นี้เป็น Component ที่ใช้สำหรับแสดงรายการหนังและส่วนการจัดการข้อมูลหนัง โดยใช้ React Hooks เพื่อจัดการสถานะข้อมูลและการแสดงผล ฟังก์ชัน useEffect ถูกใช้เพื่อดึงข้อมูลหนังเมื่อ Component ถูกโหลดเสร็จแล้ว ส่วนของการแสดงข้อมูลหนังในรูปแบบตารางถูกกำหนดโดยใช้ Ant Design Table และมีฟังก์ชันสำหรับการลบหนังและเปิด Modal สำหรับเพิ่มหรือแก้ไขข้อมูลหนัง โดยใช้ MovieForm Component ในการจัดการฟอร์มและการส่งข้อมูลไปยัง API สำหรับการเพิ่มหรือแก้ไขข้อมูล



```

1  /**
2   * Component สำหรับแสดงรายการหนังและส่วนของการจัดการข้อมูลหนัง
3   * @returns {JSX.Element} - Element ที่มีรายการหนังและส่วนการจัดการข้อมูลหนัง
4  */
5  function MoviesList() {
6    // สถานะของข้อมูลหนังและการแสดง Modal สำหรับฟอร์มหนัง
7    const [movies, setMovies] = React.useState([]);
8    const [showMovieFormModal, setShowMovieFormModal] = React.useState(false);
9    const [selectedMovie, setSelectedMovie] = React.useState(null);
10   const [formType, setFormType] = React.useState("add");
11
12   const dispatch = useDispatch();

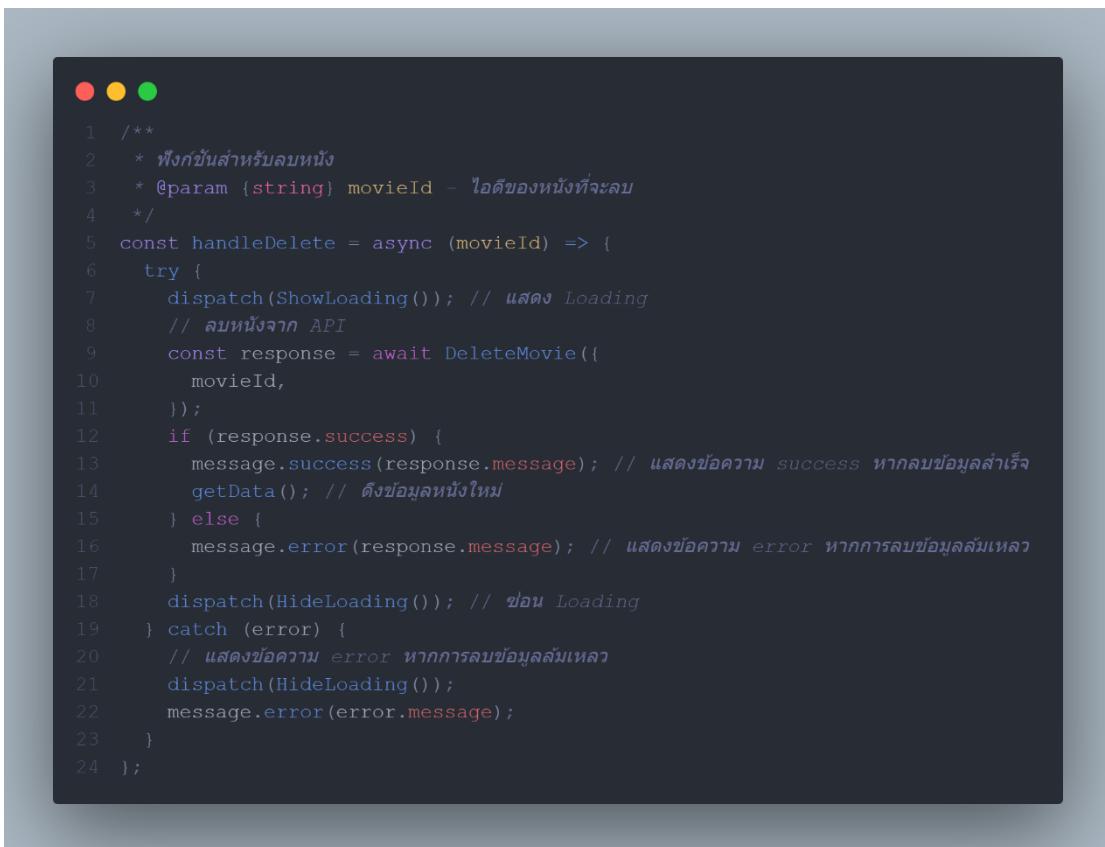
```



```

1  /**
2   * ฟังก์ชันสำหรับดึงข้อมูลหนัง
3   */
4  const getData = async () => {
5    try {
6      dispatch>ShowLoading(); // แสดง Loading
7      const response = await GetAllMovies(); // ดึงข้อมูลหนังจาก API
8      if (response.success) {
9        setMovies(response.data); // กำหนดข้อมูลหนังให้กับ state
10     } else {
11       message.error(response.message); // แสดงข้อความ error หากการดึงข้อมูลล้มเหลว
12     }
13     dispatch(HideLoading()); // ปิด Loading
14   } catch (error) {
15     // แสดงข้อความ error หากการดึงข้อมูลล้มเหลว
16     dispatch(HideLoading());
17     message.error(error.message);
18   }
19 };

```



```
1  /**
2   * ពីរក្រែងសំណងសម្របលើអត្ថបន្ទាន់
3   * @param {string} movieId - ឈើតិចខែងអត្ថបន្ទាន់ទៅលើ
4   */
5  const handleDelete = async (movieId) => {
6    try {
7      dispatch>ShowLoading(); // ផ្តល់ Loading
8      // លើអត្ថបន្ទាន់ទៅលើ API
9      const response = await DeleteMovie({
10        movieId,
11      );
12      if (response.success) {
13        message.success(response.message); // ផ្តល់ជំនួយ success តាមការលើខ្លួន
14        getData(); // ទិញដោយអត្ថបន្ទាន់ថ្មី
15      } else {
16        message.error(response.message); // ផ្តល់ជំនួយ error តាមការលើខ្លួន
17      }
18      dispatch>HideLoading(); // បន្លាក់ Loading
19    } catch (error) {
20      // ផ្តល់ជំនួយ error តាមការលើខ្លួន
21      dispatch>HideLoading();
22      message.error(error.message);
23    }
24  };
```

```
1 // គោលមនីថា ប្រព័ន្ធដែលបានបង្កើតឡើងនៅក្នុងការបង្កើត
2 const columns = [
3   {
4     title: "Poster",
5     dataIndex: "poster",
6     render: (text, record) => {
7       return (
8         <img
9           src={record.poster}
10          alt="poster"
11          height="60"
12          width="80"
13          className="br-1"
14        />
15      );
16    },
17  },
18  {
19   title: "Name",
20   dataIndex: "title",
21 },
22  {
23   title: "Description",
24   dataIndex: "description",
25 },
26  {
27   title: "Duration",
28   dataIndex: "duration",
29 },
30  {
31   title: "Genre",
32   dataIndex: "genre",
33 },
34  {
35   title: "Language",
36   dataIndex: "language",
37 },
38  {
39   title: "Release Date",
40   dataIndex: "releaseDate",
41   render: (text, record) => {
42     return moment(record.releaseDate).format("DD-MM-YYYY");
43   },
44 },
```

```
● ● ●  
1      {  
2          title: "Action",  
3          dataIndex: "action",  
4          render: (text, record) => {  
5              return (  
6                  <div className="flex gap-1">  
7                      {/* ឯកសារអនីង */}  
8                      <i  
9                          className="ri-delete-bin-line"  
10                         style={{ color: "red" }}  
11                         onClick={() => {  
12                             handleDelete(record._id);  
13                         } }  
14                     ></i>  
15                     {/* ឯកសារក្រុងខែមានឯកសារ */}  
16                     <i  
17                         className="ri-pencil-line"  
18                         style={{ color: "blue" }}  
19                         onClick={() => {  
20                             setSelectedMovie(record);  
21                             setFormType("edit");  
22                             setShowMovieFormModal(true);  
23                         } }  
24                     ></i>  
25                 </div>  
26             );  
27         },  
28     ),  
29 ];  
30  
31 // เมื่อ Component ត្រូវទាញយក នឹងធ្វើការដោយបញ្ចប់  
32 useEffect(() => {  
33     getData();  
34 }, []);
```

```
1  return (
2    <div>
3      <div className="flex justify-end mb-1">
4        {/* ປົມເພີ່ມຫນັ້ງ */}
5        <Button
6          title="Add Movie"
7          variant="outlined"
8          onClick={() => {
9            setShowMovieFormModal(true);
10           setFormType("add");
11         } }
12       />
13     </div>
14
15   {/* ດາວກາງແສດງຮາຍກາຮ່ານັ້ງ */}
16   <Table columns={columns} dataSource={movies} />
17
18   {/* Modal ສໍາໜັບເທິນທີ່ອັກໃນຂໍ້ມູນຫນັ້ງ */}
19   {showMovieFormModal && (
20     <MovieForm
21       showMovieFormModal={showMovieFormModal}
22       setShowMovieFormModal={setShowMovieFormModal}
23       selectedMovie={selectedMovie}
24       setSelectedMovie={setSelectedMovie}
25       formType={formType}
26       getData={getData}
27     />
28   ) }
29   </div>
30 );
31 }
32
33 export default MoviesList;
```

## TheatresList.js

**คำอธิบาย :** ไฟล์นี้เป็น Component ที่ใช้สำหรับแสดงรายการโรงหนังและการจัดการสถานะของโรงหนัง โดยใช้ React Hooks เพื่อจัดการสถานะข้อมูลและการแสดงผล ฟังก์ชัน useEffect ถูกใช้เพื่อดึงข้อมูลโรงหนังเมื่อ Component ถูกโหลดเสร็จแล้ว ในส่วนของการแสดงข้อมูลโรงหนังใช้ Ant Design Table และมีฟังก์ชันสำหรับการเปลี่ยนสถานะของโรงหนัง แสดงผลในรูปแบบของปุ่ม Approve หรือ Reject โดยการคลิกที่ปุ่มจะเปลี่ยนสถานะของโรงหนังจาก Active เป็น Pending หรือจาก Pending เป็น Active และทำการอัพเดทข้อมูลโรงหนังผ่าน API ซึ่งโค้ดในฟังก์ชัน handleStatusChange จะดำเนินการเรียกใช้ API ในการอัพเดทสถานะของโรงหนังที่เลือก



```

1  /**
2   * Component สำหรับแสดงรายการโรงหนังและการจัดการสถานะของโรงหนัง
3   * @returns {JSX.Element} - Element ที่มีรายการโรงหนังและส่วนการจัดการสถานะของโรงหนัง
4  */
5  function TheatresList() {
6    // สถานะของข้อมูลโรงหนังและฟังก์ชัน dispatch
7    const [theatres = [], setTheatres] = useState([]);
8    const dispatch = useDispatch();
9
10   /**
11    * ฟังก์ชันสำหรับดึงข้อมูลโรงหนัง
12    */
13   const getData = async () => {
14     try {
15       dispatch>ShowLoading(); // แสดง Loading
16       const response = await GetAllTheatres(); // ดึงข้อมูลโรงหนังจาก API
17       if (response.success) {
18         setTheatres(response.data); // กำหนดข้อมูลโรงหนังให้กับ state
19       } else {
20         message.error(response.message); // แสดงข้อความ error หากการดึงข้อมูลล้มเหลว
21       }
22       dispatch(HideLoading()); // ปิด Loading
23     } catch (error) {
24       // แสดงข้อความ error หากการดึงข้อมูลล้มเหลว
25       dispatch(HideLoading());
26       message.error(error.message);
27     }
28   };

```

```
 1  /**
 2   * ฟังก์ชันสำหรับเปลี่ยนสถานะของโรงหนัง (อนุมัติหรือปฏิเสธ)
 3   * @param {object} theatre - ข้อมูลของโรงหนัง
 4   */
 5  const handleStatusChange = async (theatre) => {
 6    try {
 7      dispatch(ShowLoading()); // แสดง Loading
 8      // เปลี่ยนสถานะของโรงหนัง
 9      const response = await UpdateTheatre({
10        theatreId: theatre._id,
11        ...theatre,
12        isActive: !theatre.isActive,
13      });
14      if (response.success) {
15        // แสดงข้อความ success หากการเปลี่ยนสถานะสำเร็จ
16        message.success(response.message);
17        getData(); // ดึงข้อมูลโรงหนังใหม่
18      } else {
19        // แสดงข้อความ error หากการเปลี่ยนสถานะล้มเหลว
20        message.error(response.message);
21      }
22      dispatch(HideLoading()); // ปิด Loading
23    } catch (error) {
24      // แสดงข้อความ error หากการเปลี่ยนสถานะล้มเหลว
25      dispatch(HideLoading());
26      message.error(error.message);
27    }
28  };

```

```
1 // គោលណ៍ទៅឱ្យសេចក្តីផ្តល់ព័ត៌មាននៃការរាយ
2 const columns = [
3     {
4         title: "Name",
5         dataIndex: "name",
6     },
7     {
8         title: "Address",
9         dataIndex: "address",
10    },
11    {
12        title: "Phone",
13        dataIndex: "phone",
14    },
15    {
16        title: "Email",
17        dataIndex: "email",
18    },
19    {
20        title: "Owner",
21        dataIndex: "owner",
22        render: (text, record) => {
23            return record.owner.name;
24        },
25    },
26    {
27        title: "Status",
28        dataIndex: "isActive",
29        render: (text, record) => {
30            if (text) {
31                return "Active";
32            } else {
33                return "Pending";
34            }
35        },
36    },

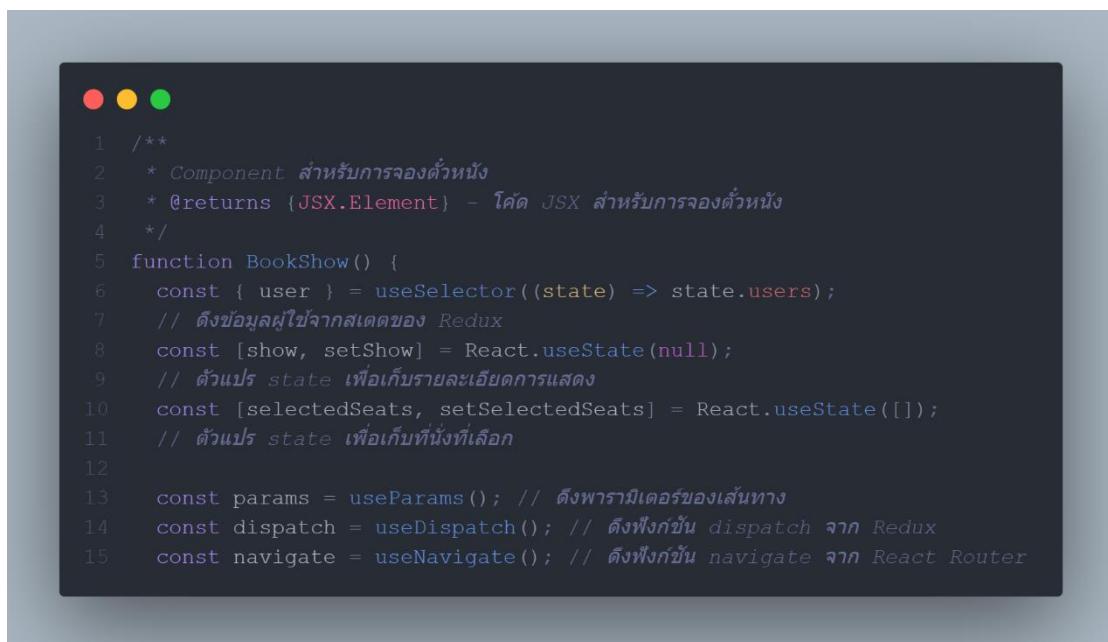
```

```
1      {
2        title: "Action",
3        dataIndex: "action",
4        render: (text, record) => {
5          return (
6            <div className="flex gap-1">
7              {/* មុនបែលីយនសាតានេខែងទេរងហួល */}
8              {record.isActive && (
9                <span
10                  className="underline"
11                  onClick={() => handleStatusChange(record)}
12                >
13                  Reject
14                </span>
15              ) }
16              {!record.isActive && (
17                <span
18                  className="underline"
19                  onClick={() => handleStatusChange(record)}
20                >
21                  Approve
22                </span>
23              ) }
24            </div>
25          );
26        },
27      },
28    ];
29
30 // เมื่อ Component ត្រូវណូលគឺសេវា នឹងតើងខ្លួនុលទេរងហួល
31 useEffect(() => {
32   getData();
33 }, []);
34
35 return (
36   <div>
37     {/* គារចាយនិងរាយការទេរងហួល */}
38     <Table columns={columns} dataSource={theatres} />
39   </div>
40 );
41 }
42
43 export default TheatresList;
```

## หน้าจอจองที่นั่ง (Seat Reservation Page)

### index.js

**คำอธิบาย :** ไฟล์นี้เป็น Component สำหรับการจองตัวหนัง โดยมีการใช้ React Hooks เพื่อจัดการสถานะข้อมูลและการแสดงผล ในส่วนของการแสดงที่นั่ง ใช้การรวมคอลัมน์และแถวเพื่อแสดงที่นั่งที่ว่างและที่ถูกจอง และสามารถเลือกที่นั่งได้ผ่านการคลิก ซึ่งที่นั่งที่ถูกเลือกจะถูกเน้นด้วยสีและสามารถยกเลิกการเลือกได้ด้วยการคลิกอีกครั้ง ในส่วนของการจ่ายเงิน ใช้ Stripe Checkout เพื่อทำการจ่ายเงินผ่านบัตรเครดิต โดยสามารถกรอกข้อมูลการจ่ายเงินและยืนยันการจ่ายเงินได้ หลังจากทำการจ่ายเงินเสร็จสิ้น จะทำการจองตัวโดยส่งข้อมูลการจองไปยัง API BookShowTickets และเมื่อการจองสำเร็จจะทำการแสดงข้อความสำเร็จและเปลี่ยนเส้นทางไปยังหน้าโปรดีของผู้ใช้งาน โดยสามารถกลับไปยังหน้าหลักได้ด้วยการคลิกที่ปุ่ม "Back to Home".



```

1  /**
2   * Component สำหรับการจองตัวหนัง
3   * @returns {JSX.Element} - โคด JSX สำหรับการจองตัวหนัง
4  */
5 function BookShow() {
6   const { user } = useSelector((state) => state.users);
7   // ดึงข้อมูลผู้ใช้จากสเกตช์ของ Redux
8   const [show, setShow] = React.useState(null);
9   // ตัวแปร state เพื่อเก็บรายละเอียดการแสดง
10  const [selectedSeats, setSelectedSeats] = React.useState([]);
11  // ตัวแปร state เพื่อเก็บที่นั่งที่เลือก
12
13  const params = useParams(); // ดึงพารามิเตอร์ของเส้นทาง
14  const dispatch = useDispatch(); // ดึงฟังก์ชัน dispatch จาก Redux
15  const navigate = useNavigate(); // ดึงฟังก์ชัน navigate จาก React Router

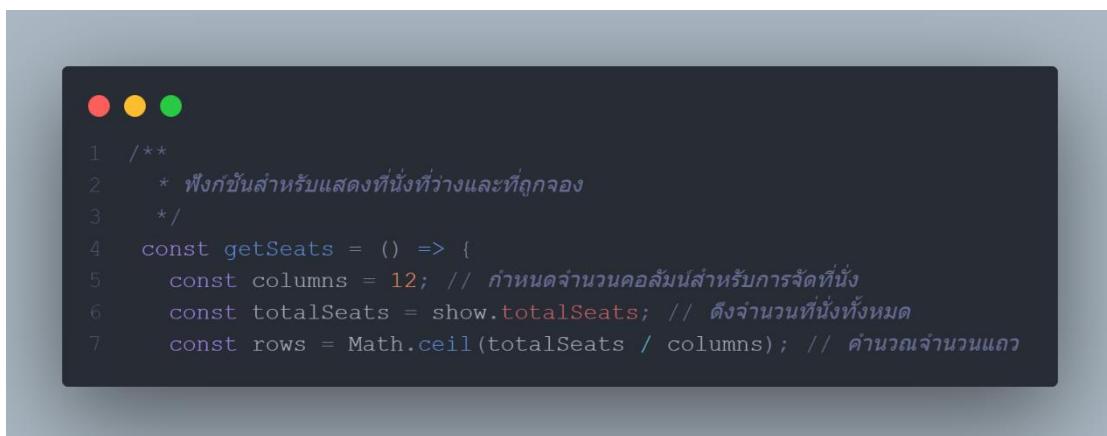
```



```

1  /**
2   * ฟังก์ชันสำหรับดึงข้อมูลการแสดงตัวอย่าง
3   */
4  const getData = async () => {
5    try {
6      dispatch>ShowLoading();
7      const response = await GetShowById({ // เรียกใช้งาน GetShowById API function
8        showId: params.id, // ส่ง showId เป็นพารามิเตอร์
9      });
10     if (response.success) {
11       setShow(response.data); // กำหนดรายละเอียดการแสดงให้กับ state
12     } else {
13       message.error(response.message); // แสดงข้อความผิดพลาด
14     }
15     dispatch>HideLoading();
16   } catch (error) {
17     dispatch>HideLoading();
18     message.error(error.message);
19   }
20 };

```



```

1  /**
2   * ฟังก์ชันสำหรับแสดงที่นั่งที่ว่างและที่ถูกจอง
3   */
4  const getSeats = () => {
5    const columns = 12; // กำหนดจำนวนคอลัมน์สำหรับการจัดที่นั่ง
6    const totalSeats = show.totalSeats; // ดึงจำนวนที่นั่งทั้งหมด
7    const rows = Math.ceil(totalSeats / columns); // คำนวณจำนวนแถว

```



```

1 return (
2   <div className="flex gap-1 flex-col p-2 card">
3     {Array.from(Array(rows).keys()).map((seat, index) => { // วนลูปผ่านแต่ละแถว
4       return (
5         <div className="flex gap-1 justify-center">
6           {Array.from(Array(columns).keys()).map((column, index) => { // วนลูปผ่านคอลัมน์
7             const seatNumber = seat * columns + column + 1; // คำนวณหมายเลขที่นั่ง
8             let seatClass = "seat"; // กำหนดคลาสที่นั่งเริ่มต้น
9
10            if (selectedSeats.includes(seat * columns + column + 1)) { // ตรวจสอบว่าที่นั่งถูกเลือกหรือไม่
11              seatClass = seatClass + " selected-seat"; // เพิ่มคลาสที่มีชื่อที่ถูกเลือก
12            }
13
14            if (show.bookedSeats.includes(seat * columns + column + 1)) { // ตรวจสอบว่าที่นั่งถูกจองหรือไม่
15              seatClass = seatClass + " booked-seat"; // เพิ่มคลาสที่มีชื่อที่ถูกจอง
16            }
17          )
18        );
19      );
20    );
21  );
22);
23</div>
24);
25);

```



```

1 return (
2   seat * columns + column + 1 <= totalSeats && ( // ตรวจสอบว่าเป็นที่นั่งที่ถูกกำหนดหรือไม่
3     <div
4       className={seatClass} // กำหนดคลาสที่นั่ง
5       onClick={() => {
6         if (selectedSeats.includes(seatNumber)) { // ตรวจสอบว่าที่นั่งถูกเลือกหรือไม่
7           setSelectedSeats( // อัปเดตรายการที่นั่งที่เลือก
8             selectedSeats.filter((item) => item !== seatNumber)
9           );
10         } else {
11           setSelectedSeats([...selectedSeats, seatNumber]); // เพิ่มที่นั่งที่เลือกเข้าไปในรายการ
12         }
13       })
14     >
15       <h1 className="text-sm">(seat * columns + column + 1)</h1> // แสดงหมายเลขที่นั่ง
16     </div>
17   );
18 );
19 );
20 );
21 );
22 );
23</div>
24 );
25 );

```

```
1  /**
2   * ฟังก์ชันสำหรับจองตั๋วหนัง
3   * @param {string} transactionId - รหัสธุรกรรม
4   */
5 const book = async (transactionId) => {
6   try {
7     dispatch>ShowLoading();
8     const response = await BookShowTickets({ // เรียกใช้งาน BookShowTickets API function
9       show: params.id, // ส่ง showId เป็นพารามิเตอร์
10      seats: selectedSeats, // ส่งที่นั่งที่เลือก
11      transactionId,
12      user: user._id, // ส่ง ID ของผู้ใช้
13    });
14   if (response.success) {
15     message.success(response.message); // แสดงข้อความสำเร็จ
16     navigate("/profile"); // เปิดหน้าจอสำหรับโปรไฟล์
17   } else {
18     message.error(response.message); // แสดงข้อความผิดพลาด
19   }
20   dispatch(HideLoading()); // ส่งการกระทำเพื่อปิดหน้าจอ
21 } catch (error) {
22   message.error(error.message); // แสดงข้อความผิดพลาด
23   dispatch(HideLoading());
24 }
25 };
```



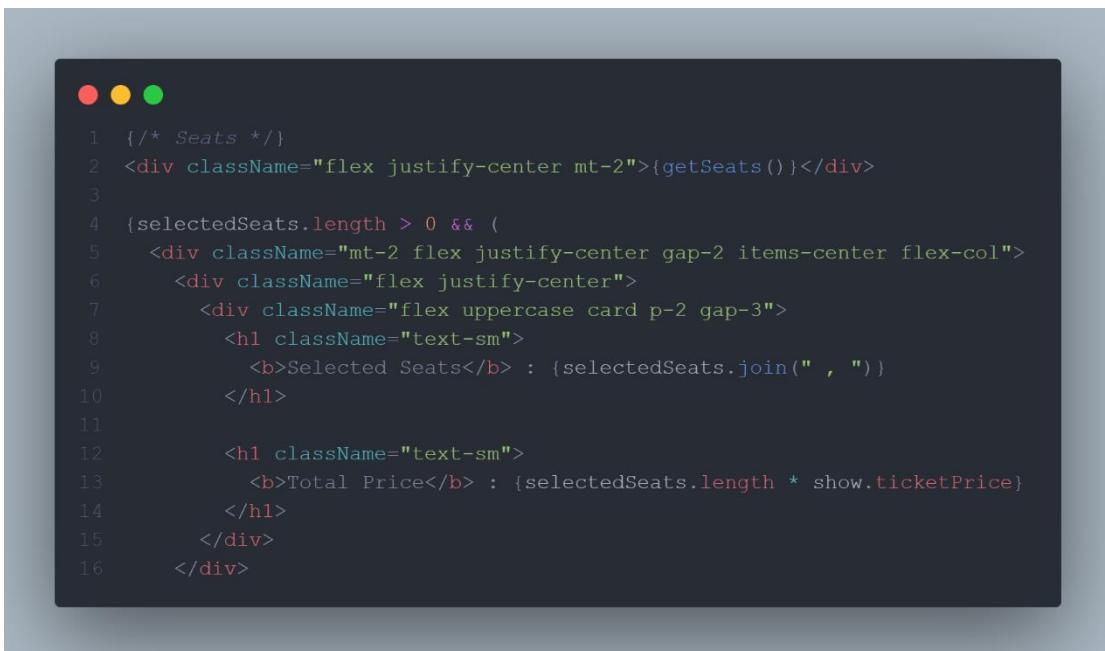
```
1  /**
2   * ฟังก์ชันสำหรับการจ่ายเงินผ่าน Stripe
3   * @param {object} token - ข้อมูล Token จากการจ่ายเงินผ่าน Stripe
4  */
5 const onToken = async (token) => {
6  try {
7    dispatch>ShowLoading();
8    const response = await MakePayment( // เรียกใช้งาน MakePayment API function
9      token,
10     selectedSeats.length * show.ticketPrice * 100 // ค่านวนเดียดเงินที่ต้องจ่าย
11   );
12   if (response.success) {
13     message.success(response.message);
14     book(response.data); // เรียกใช้ฟังก์ชันจองตั๋ว
15   } else {
16     message.error(response.message);
17   }
18   dispatch(HideLoading());
19 } catch (error) {
20   message.error(error.message);
21   dispatch(HideLoading());
22 }
23 };
```



```

1  useEffect(() => { // useEffect hook เพื่อตั้งรายละเอียดการแสดงเมื่อคุณไฟแนนซ์โหลดเสร็จ
2    getData();
3  }, []);
4
5  return (
6    show && (
7      <div>
8        {/* Show Info */}
9        <div className="flex justify-between card p-2 items-center">
10          <div>
11            <h1>(show.theatre.name)</h1>
12            <h1>(show.theatre.address)</h1>
13          </div>
14
15          <div>
16            <h1>(show.movie.title) ({show.movie.language})</h1>
17          </div>
18
19        </div>
20
21        <div>
22          <h1>(moment(show.date).format("MMM Do yyyy")) - (" ")</h1>
23          <h1>(moment(show.time, "HH:mm").format("hh:mm A"))</h1>
24        </div>
25      </div>
26    )
27  );

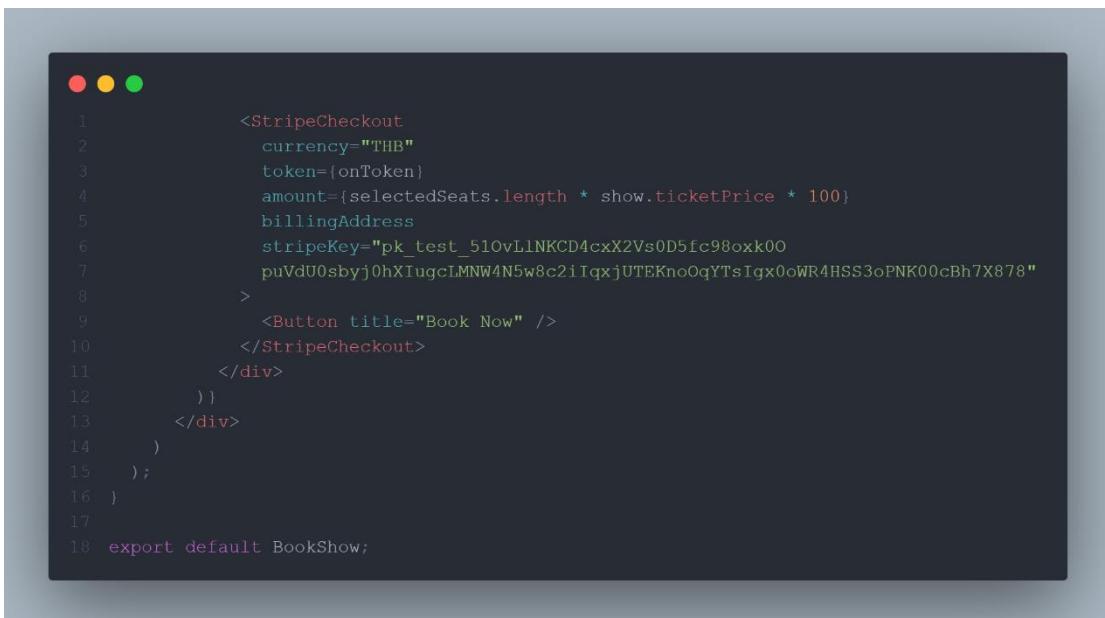
```



```

1  /* Seats */
2  <div className="flex justify-center mt-2">{getSeats()}</div>
3
4  {selectedSeats.length > 0 && (
5    <div className="mt-2 flex justify-center gap-2 items-center flex-col">
6      <div className="flex justify-center">
7        <div className="flex uppercase card p-2 gap-3">
8          <h1>Selected Seats</h1>
9          <b> : {selectedSeats.join(", ")}</b>
10         </div>
11
12        <h1>Total Price</h1>
13        <b> : {selectedSeats.length * show.ticketPrice}</b>
14      </div>
15    </div>
16  );

```



```
1      <StripeCheckout
2        currency="THB"
3        token=(onToken)
4        amount={selectedSeats.length * show.ticketPrice * 100}
5        billingAddress
6        stripeKey="pk_test_510vL1NKCD4cxX2Vs0D5fc98oxk00
7        puVdU0sbyj0hXIugcLMNW4N5w8c2iTqxjUTEKnoOqYTsIgx0oWR4HSS3oPNK00cBh7X878"
8      >
9        <Button title="Book Now" />
10      </StripeCheckout>
11    </div>
12  ) )
13 </div>
14 )
15 );
16 }
17
18 export default BookShow;
```

## หน้าหลัก (Home Page)

### index.js

**คำอธิบาย :** ไฟล์นี้เป็น Component สำหรับหน้า Home ของเว็บไซต์ โดยมีการใช้ React Hooks เพื่อจัดการสถานะข้อมูลและการแสดงผล ในส่วนของการค้นหาหนัง มีการใช้งาน input element เพื่อให้ผู้ใช้สามารถกรอกข้อความเพื่อค้นหาหนังตามชื่อได้ และในส่วนของการแสดงรายการหนัง จะแสดงรายการหนังทั้งหมดที่มีในระบบ โดยสามารถค้นหาหนังตามชื่อได้ และผู้ใช้สามารถคลิกที่รูปภาพหนังเพื่อดูรายละเอียดของหนังนั้น ๆ โดยจะทำการนำทางไปยังหน้ารายละเอียดหนัง



```

1  /**
2   * Component สำหรับหน้า Home ของเว็บไซต์
3   * @returns {JSX.Element} - ได้ต่อ JSX สำหรับหน้า Home
4  */
5  function Home() {
6    const [searchText, setSearchText] = React.useState(""); // สถานะสำหรับค้นหาชื่อความ
7    const [movies, setMovies] = React.useState([]); // สถานะสำหรับเก็บข้อมูลหนัง
8    const navigate = useNavigate(); // hook สำหรับการนำทาง
9    const dispatch = useDispatch(); // hook สำหรับการใช้งาน dispatch
10
11   /**
12    * พัฒนาสำหรับดึงข้อมูลหนังทั้งหมด
13    */
14   const getData = async () => {
15     try {
16       dispatch>ShowLoading();
17       const response = await GetAllMovies();
18       // เรียกใช้งาน API เพื่อดึงข้อมูลหนังทั้งหมด
19       if (response.success) {
20         setMovies(response.data);
21       } else {
22         message.error(response.message);
23       }
24       dispatch(HideLoading());
25     } catch (error) {
26       dispatch(HideLoading());
27       message.error(error.message);
28     }
29   };
30
31   useEffect(() => {
32     getData();
33   }, []);

```

```

1  return (
2      <div>
3          /* Input សារិបគន្តា */
4          <input
5              type="text"
6              className="search-input"
7              placeholder="Search for any movies"
8              value={searchText}
9              onChange={(e) => setSearchText(e.target.value)}
10         />
11
12         /* ផែងរាយការណ៍ */
13         <Row gutter={[20]}> className="mt-2">
14             {movies
15                 .filter((movie) =>
16                     movie.title.toLowerCase().includes(searchText.toLowerCase())
17                 )
18                 .map((movie) => (
19                     <Col span={6}>
20                         /* ផែងរាយលេខធនការនៃចំណែកទី១ */
21                         <div
22                             className="card flex flex-col gap-1 cursor-pointer mb-1"
23                             onClick={() =>
24                                 navigate(
25                                     `/movie/${movie._id}?date=${moment().format("YYYY-MM-DD")}`
26                                 )
27                             }
28                         >
29                             /* ផែងរូបភាពឪតិ៍ដែរនៃចំណែក */
30                             <img src={movie.poster} alt="" height={200} />
31
32                             <div className="flex justify-center p-1">
33                                 <h1 className="text-md uppercase">{movie.title}</h1>
34                             </div>
35                         </div>
36                     </Col>
37                 )));
38             </Row>
39         </div>
40     );
41 }
42
43
44 export default Home;

```

## หน้าเข้าสู่ระบบ (Login Page)

### index.js

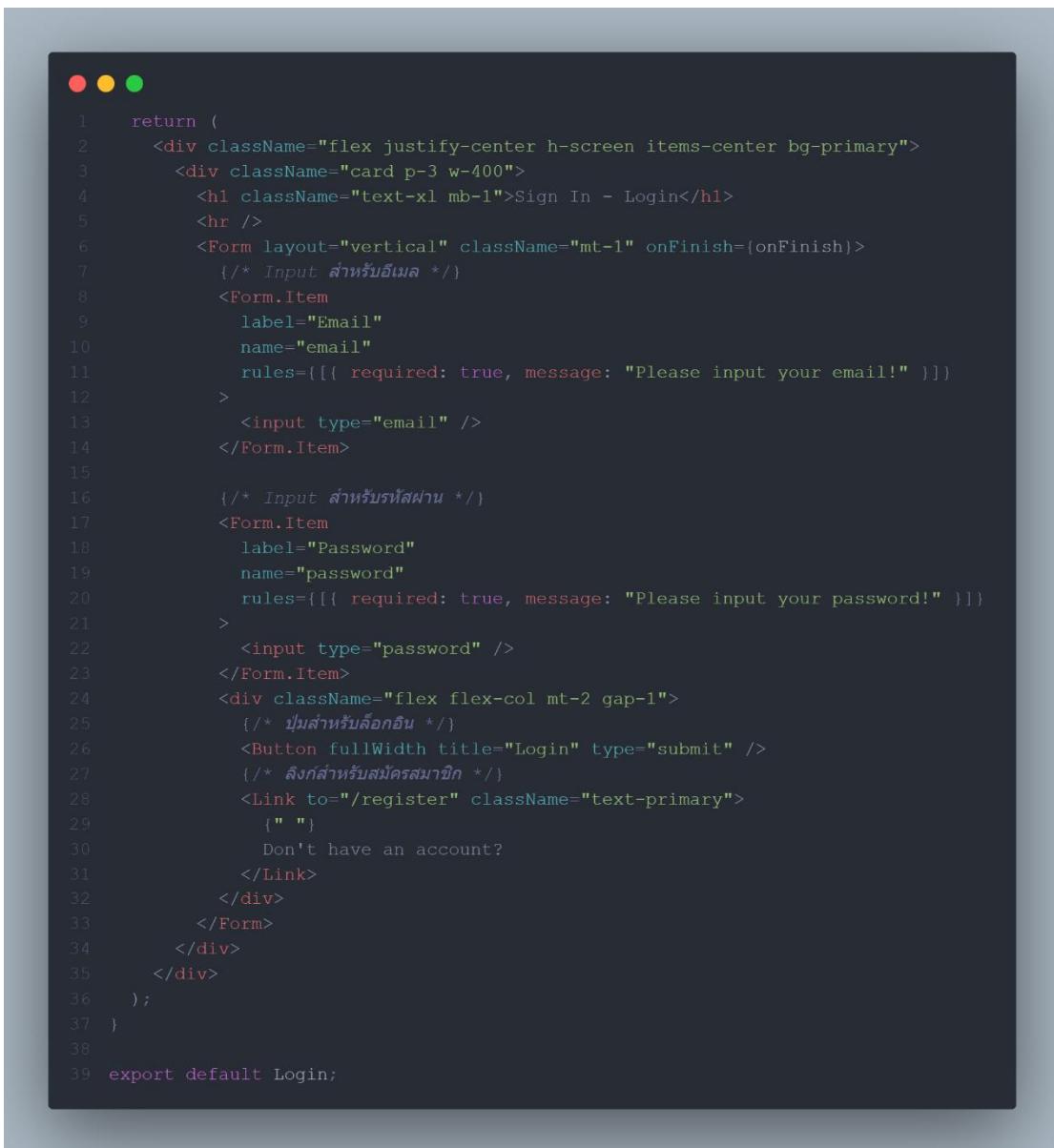
**คำอธิบาย :** ไฟล์นี้เป็น Component สำหรับหน้า Login ของเว็บไซต์ ในส่วนของการล็อกอิน ผู้ใช้จะต้องกรอกอีเมลและรหัสผ่าน เมื่อกดปุ่ม "Login" แล้วข้อมูลจะถูกส่งไปยัง API เพื่อทำการตรวจสอบ หากการล็อกอินสำเร็จ จะแสดงข้อความสำเร็จและจะเก็บ Token ที่ได้รับจาก API ลงใน Local Storage และทำการนำทางไปยังหน้าหลัก หากมี Token อยู่ใน Local Storage อยู่แล้ว ในกรณีที่ผู้ใช้มีบัญชีอยู่แล้ว สามารถคลิกที่ลิงก์ "Don't have an account?" เพื่อไปยังหน้าสมัคร สมาชิกได้ด้วยการใช้งาน hook Link ของ React Router



```

1  /**
2   * Component สำหรับหน้า Login ของเว็บไซต์
3   * @returns {JSX.Element} - โคด JSX สำหรับหน้า Login
4   */
5  function Login() {
6    const navigate = useNavigate(); // Hook สำหรับการนำทาง
7    const dispatch = useDispatch(); // Hook สำหรับการใช้งาน dispatch
8
9    /**
10     * ฟังก์ชันสำหรับการล็อกอิน
11     * @param {Object} values - ค่าที่รับมาจากการล็อกอิน
12     */
13    const onFinish = async (values) => {
14      try {
15        dispatch>ShowLoading()
16        const response = await LoginUser(values); // เรียกใช้งาน API เพื่อล็อกอิน
17        dispatch(HideLoading())
18        if (response.success) {
19          message.success(response.message);
20          localStorage.setItem("token", response.data); // เก็บ Token ลงใน Local Storage
21          window.location.href = "/";
22        } else {
23          message.error(response.message);
24        }
25      } catch (error) {
26        dispatch(HideLoading())
27        message.error(error.message);
28      }
29    };
30
31    useEffect(() => {
32      if (localStorage.getItem("token")) {
33        navigate(""); // นำทางไปยังหน้าหลัก หากมี Token อยู่ใน Local Storage
34      }
35    }, []);

```



```
1  return (
2      <div className="flex justify-center h-screen items-center bg-primary">
3          <div className="card p-3 w-400">
4              <h1 className="text-xl mb-1">Sign In - Login</h1>
5              <hr />
6              <Form layout="vertical" className="mt-1" onFinish={onFinish}>
7                  {/* Input សារីបីមេល */}
8                  <Form.Item
9                      label="Email"
10                     name="email"
11                     rules={[{ required: true, message: "Please input your email!" }]}
12                  >
13                      <input type="email" />
14                  </Form.Item>
15
16                  {/* Input សារីបីអត្ថលេខាមួយ */}
17                  <Form.Item
18                      label="Password"
19                     name="password"
20                     rules={[{ required: true, message: "Please input your password!" }]}
21                  >
22                      <input type="password" />
23                  </Form.Item>
24              <div className="flex flex-col mt-2 gap-1">
25                  {/* បុរាណសំណង់ */}
26                  <Button fullWidth title="Login" type="submit" />
27                  {/* ចូលកសារីបីសម្រាប់ចូលកសារីបី */}
28                  <Link to="/register" className="text-primary">
29                      {" "}
30                      Don't have an account?
31                  </Link>
32              </div>
33          </Form>
34      </div>
35  </div>
36 );
37 }
38
39 export default Login;
```

## โปรไฟล์ (Profile)

### Bookings.js

**คำอธิบาย :** ไฟล์นี้เป็น Component สำหรับการแสดงรายการการจองของผู้ใช้ โดยจะดึงข้อมูลการจองของผู้ใช้จาก API เมื่อคอมโพเนนต์โหลดเสร็จ และแสดงข้อมูลการจองแต่ละรายการในรูปแบบการ์ด โดยแสดงรายละเอียดของการจอง เช่น ชื่อหนัง ภาษา ชื่อโรงหนัง ที่อยู่ของโรงหนัง วันและเวลา ราคาตั๋ว หมายเลขอการจอง และรายละเอียดที่นั่ง พร้อมทั้งแสดงรูปภาพโปสเตอร์ของหนังที่จอง



```

1  /**
2   * Component สำหรับการแสดงรายการการจองของผู้ใช้
3   * @returns {JSX.Element} - คือ JSX สำหรับหน้า Bookings
4  */
5  function Bookings() {
6    const [bookings, setBookings] = useState([]);
7    // สร้าง state สำหรับเก็บข้อมูลการจอง
8    const dispatch = useDispatch(); // Hook สำหรับใช้งาน dispatch
9    const navigate = useNavigate(); // Hook สำหรับการนำทาง
10
11   /**
12    * พิ้งกี้ชั้นสำหรับการดึงข้อมูลการจองของผู้ใช้
13    */
14   const getData = async () => {
15     try {
16       dispatch>ShowLoading();
17       const response = await GetBookingsOfUser();
18       // เรียกใช้งาน API เพื่อดึงข้อมูลการจอง
19       if (response.success) {
20         setBookings(response.data);
21       } else {
22         message.error(response.message);
23       }
24       dispatch(HideLoading());
25     } catch (error) {
26       dispatch(HideLoading());
27       message.error(error.message);
28     }
29   };
30
31   useEffect(() => {
32     getData(); // เรียกใช้งานพิ้งกี้ชั้นเมื่อคอมโพเนนต์โหลด
33   }, []);

```

```
1  return (
2      <div>
3          <Row gutter={[16, 16]}>
4              {bookings.map((booking) => (
5                  <Col span={12}>
6                      <div className="card p-2 flex justify-between uppercase">
7                          <div>
8
9                              <h1 className="text-xl">
10                                 {booking.show.movie.title} ({booking.show.movie.language})
11                             </h1>
12                             <div className="divider"></div>
13                             <h1 className="text-sm">
14                                 {booking.show.theatre.name} ({booking.show.theatre.address})
15                             </h1>
16                             <h1 className="text-sm">
17                                 Date & Time: {moment(booking.show.date).format("Do MMM YYYY")} (" ")
18                                     - {moment(booking.show.time, "HH:mm").format("hh:mm A")}
19                             </h1>
20
21                             <h1 className="text-sm">
22                                 Amount : ₦ {booking.show.ticketPrice * booking.seats.length}
23                             </h1>
24                             <h1 className="text-sm">Booking ID: {booking._id}</h1>
25                         </div>
26
27                         <div>
28                             <img
29                                 src={booking.show.movie.poster}
30                                 alt=""
31                                 height=(100)
32                                 width=(100)
33                                 className="br-1"
34                             />
35                             <h1 className="text-sm">Seats: {booking.seats.join(", ")}</h1>
36                         </div>
37                     </div>
38                 </Col>
39             )));
40         </Row>
41     </div>
42 );
43 }
44
45 export default Bookings;
```

## index.js

คำอธิบาย : ไฟล์นี้เป็น Component สำหรับหน้าโปรไฟล์ของผู้ใช้ ซึ่งแสดงข้อมูลการจองของผู้ใช้ในแท็บ "Bookings" โดยใช้ Ant Design เพื่อสร้างแท็บและการจัดรูปแบบให้มีการเลือกแท็บได้ง่าย และแสดงข้อมูลการจองด้วย Component Bookings ที่เรียกใช้งานด้านใน



```
 1  /**
 2   * Component สำหรับหน้าโปรไฟล์ของผู้ใช้
 3   * @returns {JSX.Element} - โค้ด JSX สำหรับหน้าโปรไฟล์
 4   */
 5  function Profile() {
 6    return (
 7      <div>
 8        <PageTitle title="Profile" /> {/* แสดงชื่อหน้าโปรไฟล์ */}
 9
10        <Tabs defaultActiveKey="1">
11          <Tabs.TabPane tab="Bookings" key="1">
12            <Bookings />
13          </Tabs.TabPane>
14
15        </Tabs>
16      </div>
17    );
18  }
19
20 export default Profile;
```

## สมัครใช้งาน (Register)

### index.js

**คำอธิบาย :** ไฟล์นี้เป็น Component สำหรับหน้าที่ใช้ในการลงทะเบียนผู้ใช้ใหม่ ซึ่งมีการใช้ Ant Design เพื่อสร้างฟอร์มการลงทะเบียนและปุ่ม REGISTER และลิงก์สำหรับเข้าสู่ระบบหากมีบัญชีผู้ใช้แล้ว โดยมีการใช้งาน API เพื่อส่งข้อมูลการลงทะเบียนผู้ใช้ใหม่ และเมื่อลงทะเบียนสำเร็จจะเปลี่ยนเส้นทางไปยังหน้าล็อกอินอัตโนมัติ และหากมี Token ใน Local Storage ก็จะเปลี่ยนเส้นทางไปยังหน้าหลักโดยอัตโนมัติด้วย



```

1  /**
2   * Register component สำหรับหน้าที่ใช้ในการลงทะเบียนผู้ใช้ใหม่
3   * @returns {JSX.Element} - ได้ด้วย JSX สำหรับหน้าลงทะเบียน
4   */
5  function Register() {
6    // เรียกใช้ dispatch จาก Redux เพื่อใช้ในการส่ง action
7    const dispatch = useDispatch();
8
9    // เรียกใช้ hook useNavigate จาก React Router เพื่อใช้ในการเปลี่ยนเส้นทาง
10   const navigate = useNavigate();
11
12   /**
13    * เมื่อลงทะเบียนสำเร็จ ให้ส่งข้อมูลไปยัง API เพื่อลงทะเบียนผู้ใช้
14    * @param {Object} values - ข้อมูลที่ได้จากฟอร์มการลงทะเบียน
15    */
16   const onFinish = async (values) => {
17     try {
18       dispatch>ShowLoading(); // แสดง Loader
19       const response = await RegisterUser(values); // ส่งข้อมูลไปยัง API เพื่อลงทะเบียนผู้ใช้
20       dispatch(HideLoading()); // ปิด Loader เมื่อกระบวนการสำเร็จหรือเกิดข้อผิดพลาด
21       if (response.success) { // ตรวจสอบว่าการลงทะเบียนสำเร็จหรือไม่
22         message.success(response.message); // แสดงข้อความสำเร็จ
23         navigate("/login"); // เปิดเส้นทางไปยังหน้าล็อกอิน
24       } else {
25         message.error(response.message); // แสดงข้อความผิดพลาด
26       }
27     } catch (error) {
28       dispatch(HideLoading()); // ปิด Loader เมื่อเกิดข้อผิดพลาด
29       message.error(error.message); // แสดงข้อความผิดพลาด
30     }
31   };

```

```

1 // เมื่อคอมไพล์โค้ด
2 useEffect(() => {
3     if (localStorage.getItem("token")) {
4         // ถ้ามี token ใน localStorage ให้เปลี่ยนเส้นทางไปยังหน้าหลัก
5         navigate("/");
6     }
7 }, []);
8
9 // สร้าง Element ของหน้าลงทะเบียนกลับบ้อง
10 return (
11     <div className="flex justify-center h-screen items-center bg-primary">
12         <div className="card p-3 w-400">
13             <h1 className="text-xl mb-1">Sign Up - Register</h1>
14             <hr />
15             <Form layout="vertical" className="mt-1" onFinish={onFinish}>
16                 <Form.Item
17                     label="Username"
18                     name="name"
19                     rules={[{ required: true, message: "Please input your username!" }]}
20                 >
21                     <input type="text" />
22                 </Form.Item>
23                 <Form.Item
24                     label="Email"
25                     name="email"
26                     rules={[{ required: true, message: "Please input your email!" }]}
27                 >
28                     <input type="email" />
29                 </Form.Item>
30                 <Form.Item
31                     label="Password"
32                     name="password"
33                     rules={[{ required: true, message: "Please input your password!" }]}
34                 >
35                     <input type="password" />
36                 </Form.Item>
37                 <div className="flex flex-col mt-2 gap-1">
38                     <Button fullWidth title="REGISTER" type="submit" />
39                     <Link to="/login" className="text-primary">
40                         {" "}
41                         Already have an account?
42                     </Link>
43                 </div>
44             </Form>
45         </div>
46     </div>
47 );
48 }
49
50 export default Register;

```

## รายการการแสดง (Show Listings)

### index.js

**คำอธิบาย :** ไฟล์ Shows ใช้สำหรับแสดงและจัดการรายการฉายภาพยนตร์ของโรงหนัง โดยมีฟังก์ชันเพื่อดึงข้อมูลการฉายภาพยนตร์ของโรงหนังจาก API และเพิ่มหรือลบการฉายภาพยนตร์ ตลอดจนการใช้งาน Component ต่าง ๆ ของ Ant Design เช่น Modal, Table, Form, และ Button เพื่อให้ผู้ใช้สามารถดูข้อมูลและจัดการรายการฉายภาพยนตร์ได้อย่างมีประสิทธิภาพ



```

1  /**
2   * Shows component สำหรับแสดงและจัดการรายการฉายภาพยนตร์ของโรงหนัง
3   * @param {Object} props - ข้อมูลและฟังก์ชันที่ใช้ในการจัดการรายการฉายภาพยนตร์
4   * @returns {JSX.Element} - ได้ JSX สำหรับแสดงรายการฉายภาพยนตร์ของโรงหนัง
5  */
6  function Shows({ openShowsModal, setOpenShowsModal, theatre }) {
7    const [view, setView] = React.useState("table");
8    const [shows, setShows] = React.useState([]);
9    const [movies, setMovies] = React.useState([]);
10
11   const dispatch = useDispatch();
12
13   const getData = async () => {
14     try {
15       dispatch>ShowLoading();
16       const moviesResponse = await GetAllMovies();
17       if (moviesResponse.success) {
18         setMovies(moviesResponse.data);
19       } else {
20         message.error(moviesResponse.message);
21       }
22
23       const showsResponse = await GetAllShowsByTheatre({
24         theatreId: theatre._id,
25       });
26       if (showsResponse.success) {
27         setShows(showsResponse.data);
28       } else {
29         message.error(showsResponse.message);
30       }
31
32       dispatch(HideLoading());
33     } catch (error) {
34       message.error(error.message);
35       dispatch(HideLoading());
36     }
37   };

```

```
1 const handleAddShow = async (values) => {
2   try {
3     dispatch>ShowLoading();
4     const response = await AddShow({
5       ...values,
6       theatre: theatre._id,
7     });
8     if (response.success) {
9       message.success(response.message);
10    getData();
11    setView("table");
12  } else {
13    message.error(response.message);
14  }
15  dispatch>HideLoading();
16 } catch (error) {
17   message.error(error.message);
18  dispatch>HideLoading();
19 }
20 };
21
22 const handleDelete = async (id) => {
23   try {
24     dispatch>ShowLoading();
25     const response = await DeleteShow({ showId: id });
26
27     if (response.success) {
28       message.success(response.message);
29       getData();
30     } else {
31       message.error(response.message);
32     }
33     dispatch>HideLoading();
34 } catch (error) {
35   message.error(error.message);
36  dispatch>HideLoading();
37 }
38 };
```

```
 1 const columns = [
 2   {
 3     title: "Show Name",
 4     dataIndex: "name",
 5   },
 6   {
 7     title: "Date",
 8     dataIndex: "date",
 9     render: (text, record) => {
10       return moment(text).format("Do MMM YYYY");
11     },
12   },
13   {
14     title: "Time",
15     dataIndex: "time",
16   },
17   {
18     title: "Movie",
19     dataIndex: "movie",
20     render: (text, record) => {
21       return record.movie.title;
22     },
23   },
24   {
25     title: "Ticket Price",
26     dataIndex: "ticketPrice",
27   },
28   {
29     title: "Total Seats",
30     dataIndex: "totalSeats",
31   },
32   {
33     title: "Available Seats",
34     dataIndex: "availableSeats",
35     render: (text, record) => {
36       return record.totalSeats - record.bookedSeats.length;
37     },
38   },
39   {
40     title: "Action",
41     dataIndex: "action",
42     render: (text, record) => {
43       return (
44         <div className="flex gap-1 items-center">
45           {record.bookedSeats.length === 0 && (
46             <i
47               className="ri-delete-bin-line"
48               onClick={() => {
49                 handleDelete(record._id);
50               }}
51             ></i>
52           )}
53         </div>
54       );
55     },
56   },
57 ];
```

```
● ● ●  
1  useEffect(() => {  
2    getData();  
3  }, []);  
4  
5  return (  
6    <Modal  
7      title=""  
8      open={openShowsModal}  
9      onCancel={() => setOpenShowsModal(false)}  
10     width={1400}  
11     footer={null}  
12    >  
13    <h1 className="text-primary text-md uppercase mb-1">  
14      Theatre : {theatre.name}  
15    </h1>  
16    <hr />
```

```
● ● ●  
1  <div className="flex justify-between mt-1 mb-1 items-center">  
2    <h1 className="text-md uppercase">  
3      {view === "table" ? "Theatres Info" : "New showtime information"}  
4    </h1>  
5    {view === "table" && (  
6      <Button  
7        variant="outlined"  
8        title="Add Movie"  
9        onClick={() => {  
10          setView("form");  
11        }}  
12      />  
13    )}  
14  </div>  
15  {view === "table" && <Table columns={columns} dataSource={shows} />}
```

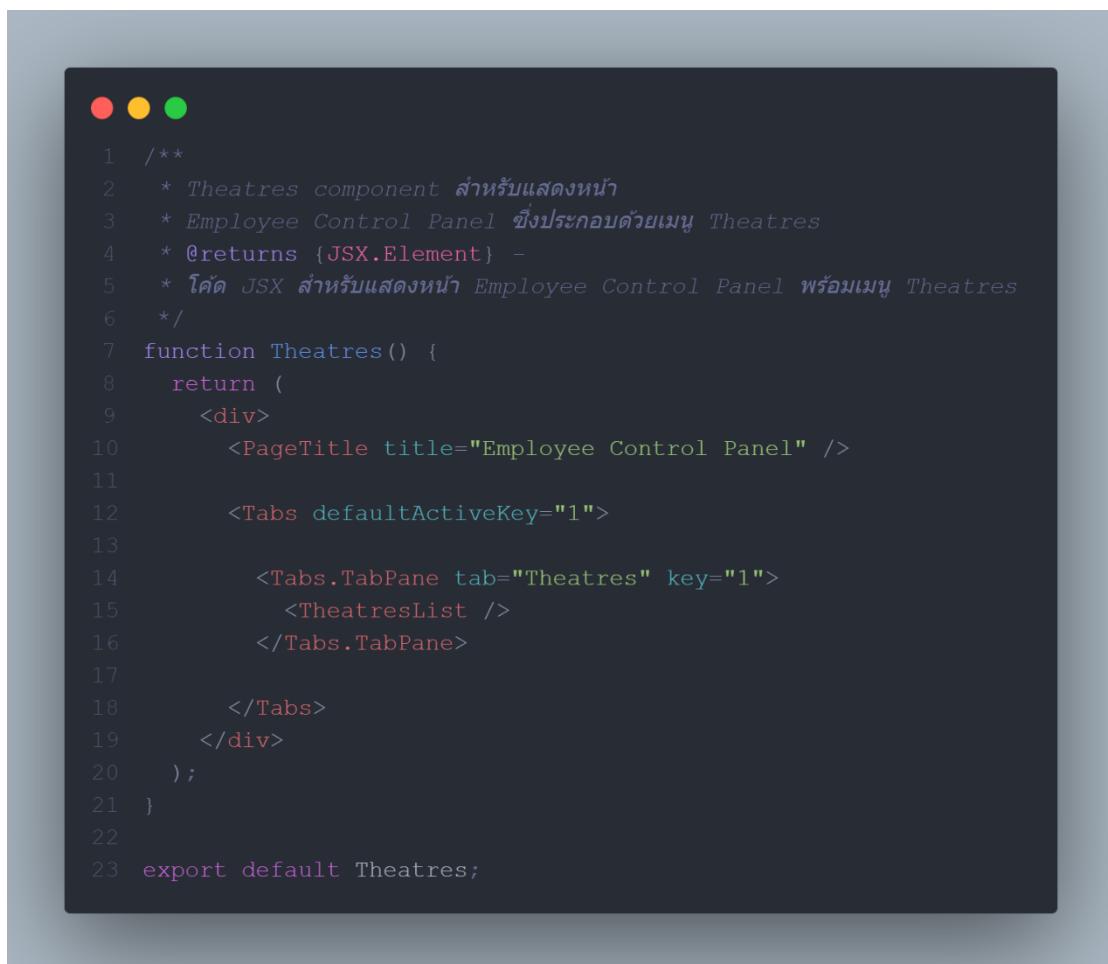
```
● ● ●  
1  {view === "form" && (  
2    <Form layout="vertical" onFinish={handleAddShow}>  
3      <Row gutter={[16, 16]}>  
4        <Col span={8}>  
5          <Form.Item  
6            label="Theatres"  
7            name="name"  
8            rules={[{ required: true, message: "Please input theatres name!" }]}  
9          >  
10         <input />  
11       </Form.Item>  
12     </Col>  
13     <Col span={8}>  
14       <Form.Item  
15         label="Date"  
16         name="date"  
17         rules={[{ required: true, message: "Please input available date!" }]}  
18       >  
19       <input  
20         type="date"  
21         min={new Date().toISOString().split("T")[0]}  
22       />  
23     </Form.Item>  
24   </Col>  
25  
26   <Col span={8}>  
27     <Form.Item  
28       label="Time"  
29       name="time"  
30       rules={[{ required: true, message: "Please input showtime!" }]}  
31     >  
32     <input type="time" />  
33   </Form.Item>  
34 </Col>  
35  
36 <Col span={8}>  
37   <Form.Item  
38     label="Movie"  
39     name="movie"  
40     rules={[{ required: true, message: "Please select any movie!" }]}  
41   >  
42   <select>  
43     <option value="">Select Movie</option>  
44     {movies.map((movie) => (  
45       <option value={movie._id}>{movie.title}</option>  
46     )))  
47   </select>  
48 </Form.Item>  
49 </Col>
```

```
1      <Col span={8}>
2          <Form.Item
3              label="Ticket Price"
4              name="ticketPrice"
5              rules={[
6                  { required: true, message: "Please select ticket price!" },
7              ]}
8          >
9              {/* <input type="number" /> */}
10             <select name="" id=""
11                 >
12                 <option value="">Select Price</option>
13                 <option value="220">220</option>
14                 <option value="240">240</option>
15                 <option value="300">300</option>
16             </select>
17         </Form.Item>
18     </Col>
19
20     <Col span={8}>
21         <Form.Item
22             label="Total Seats"
23             name="totalSeats"
24             rules={[
25                 { required: true, message: "Please select total seats!" },
26             ]}
27         >
28             {/* <input type="number" /> */}
29             <select name="" id=""
30                 >
31                 <option value="">Total Seats</option>
32                 <option value="60">60</option>
33                 <option value="72">72</option>
34                 <option value="84">84</option>
35             </select>
36         </Form.Item>
37     </Col>
38 </Row>
39
40     <div className="flex justify-end gap-1">
41         <Button
42             variant="outlined"
43             title="Cancel"
44             onClick={() => {
45                 setView("table");
46             }}
47         />
48         <Button variant="contained" title="SAVE" type="submit" />
49     </div>
50 </Form>
51     ) )
52 </Modal>
53
54 export default Shows;
```

## สถานที่แสดง (Theatres)

### index.js

**คำอธิบาย :** ไฟล์นี้เป็นส่วนหนึ่งของ Employee Control Panel ซึ่งช่วยให้พนักงานสามารถจัดการข้อมูลโรงหนังได้อย่างมีประสิทธิภาพ ผ่านการแบ่งหน้าเป็นส่วนๆ เช่น TheatresList เพื่อแสดงรายการโรงหนังทั้งหมด และใช้งาน Tabs เพื่อเปิดหน้าจอ Theatres



```

1  /**
2   * Theatres component สำหรับแสดงหน้า
3   * Employee Control Panel ซึ่งประกอบด้วยเมนู Theatres
4   * @returns {JSX.Element} -
5   * ได้ด้วย JSX สำหรับแสดงหน้า Employee Control Panel พร้อมเมนู Theatres
6   */
7  function Theatres() {
8      return (
9          <div>
10         <PageTitle title="Employee Control Panel" />
11
12         <Tabs defaultActiveKey="1">
13
14             <Tabs.TabPane tab="Theatres" key="1">
15                 <TheatresList />
16             </Tabs.TabPane>
17
18         </Tabs>
19     </div>
20 );
21 }
22
23 export default Theatres;

```

## TheatreForm.js

คำอธิบาย : ไฟล์นี้เป็นคอมโพเนนต์สำหรับเพิ่มหรือแก้ไขข้อมูลโรงหนัง มีฟอร์มและ Modal แสดงข้อมูลโรงหนัง สามารถเรียกใช้งานผ่าน props และ getData ฟังก์ชัน onFinish ใช้สำหรับส่งข้อมูลฟอร์มไปยัง API และแสดงผลลัพธ์การดำเนินการผ่านข้อความ



```

1  /**
2   * ฟังก์ชันสำหรับการส่งข้อมูลฟอร์มและบันทึกหรือแก้ไขข้อมูลโรงหนัง
3   * @param {Object} values - ข้อมูลที่ได้จากฟอร์ม
4  */
5 const onFinish = async (values) => {
6   values.owner = user._id;
7   try {
8     dispatch>ShowLoading();
9     let response = null;
10    if (formType === "add") {
11      response = await AddTheatre(values);
12    } else {
13      values.theatreId = selectedTheatre._id;
14      response = await UpdateTheatre(values);
15    }
16
17    if (response.success) {
18      message.success(response.message);
19      setShowTheatreFormModal(false);
20      setSelectedTheatre(null);
21      getData();
22    } else {
23      message.error(response.message);
24    }
25
26    dispatch(HideLoading());
27  } catch (error) {
28    dispatch(HideLoading());
29    message.error(error.message);
30  }
31};
32

```

```
1  return (
2    <Modal
3      title={formType === "add" ? "Add Theatre" : "Edit Theatre"}
4      open={showTheatreFormModal}
5      onCancel={() => {
6        setShowTheatreFormModal(false);
7        setSelectedTheatre(null);
8      }}
9      footer={null}
10   >
11     <Form
12       layout="vertical"
13       onFinish={onFinish}
14       initialValues={selectedTheatre}
15     >
16       <Form.Item
17         label="Name"
18         name="name"
19         rules={[{ required: true,
20           message: "Please input theatre name!" }]}
21       >
22         <input type="text" />
23       </Form.Item>
24
25       <Form.Item
26         label="Address"
27         name="address"
28         rules={[{ required: true,
29           message: "Please input theatre address!" }]}
30       >
31         <textarea type="text" />
32       </Form.Item>
33
34       <Form.Item
35         label="Phone Number"
36         name="phone"
37         rules={[
38           { required: true,
39             message: "Please input theatre phone number!" },
40         ]}
41       >
42         <input type="text" />
43       </Form.Item>
```

```
1      <Form.Item  
2        label="Email"  
3        name="email"  
4        rules={[{ required: true,  
5          message: "Please input owner theatre email!" }]}  
6      >  
7        <input type="email" />  
8      </Form.Item>  
9      <div className="flex justify-end gap-1">  
10        <Button  
11          title="Cancel"  
12          variant="outlined"  
13          type="button"  
14          onClick={() => {  
15            setShowTheatreFormModal(false);  
16            setSelectedTheatre(null);  
17          }}  
18        />  
19        <Button title="Save" type="submit" />  
20      </div>  
21    </Form>  
22  </Modal>  
23 );  
24 }  
25  
26 export default TheatreForm;
```

## TheatresList.js

**คำอธิบาย :** ไฟล์นี้เป็นคอมโพเนนต์ที่แสดงรายการโรงหนังทั้งหมดและให้การจัดการโรงหนัง เมื่อโหลดคอมโพเนนต์ เรียกใช้ API เพื่อดึงข้อมูลโรงหนังของผู้ใช้ปัจจุบัน สามารถเพิ่มหรือแก้ไขข้อมูลโรงหนัง ลบโรงหนัง และดูรายการการแสดงของโรงหนังได้ ผ่านการคลิกที่ไอคอนที่เหมาะสมในตารางแสดงข้อมูลโรงหนัง และใช้งาน TheatreForm และ Shows ในการแสดงและจัดการฟอร์มและรายการการแสดงตามลำดับ



```

1  /**
2   * TheatreList component สำหรับแสดงรายการโรงหนังทั้งหมดและจัดการโรงหนัง
3   * @returns {JSX.Element} - ผลลัพธ์ JSX สำหรับแสดงและจัดการรายการโรงหนัง
4  */
5  function TheatreList() {
6    const [user] = useSelector((state) => state.users);
7    // ดึงข้อมูลผู้ใช้ปัจจุบันจาก Redux store
8    const [showTheatreFormModal = false, setShowTheatreFormModal] =
9      // สถานะการแสดง Modal ของฟอร์มโรงหนัง
10     useState(false);
11    const [selectedTheatre = null, setSelectedTheatre] = useState(null);
12    // ข้อมูลโรงหนังที่ถูกเลือกสำหรับแก้ไข
13    const [formType = "add", setFormType] = useState("add");
14    // ประเภทของฟอร์ม (เพิ่มหรือแก้ไข)
15    const [theatres = [], setTheatres] = useState([]);
16    // ข้อมูลโรงหนังทั้งหมด
17
18    const [openShowsModal = false, setOpenShowsModal] = useState(false);
19    // สถานะการแสดง Modal ของรายการการแสดง
20    const dispatch = useDispatch();
21    // เรียกใช้ dispatch เพื่อส่ง action ไปยัง Redux store
22    const navigate = useNavigate();
23    // เรียกใช้ hook ในการเปลี่ยนเส้นทาง

```

```

1 // ພຶກຂັ້ນສ່າຫວັບເດີງຂໍອມູລໂຮງໝັ້ນ
2 const getData = async () => {
3   try {
4     dispatch>ShowLoading();
5     const response = await GetAllTheatresByOwner({
6       owner: user._id, // ເຮັດກໃຫ້ງານ API ເພື່ອເຫັນຂໍອມູລໂຮງໝັ້ນໄດ້ໂດຍໃຫ້ ID ຂອງຜູ້ໃນປົວຈຸນັນ
7     });
8     if (response.success) {
9       setTheatres(response.data); // ເຫັດຂໍອມູລໂຮງໝັ້ນທີ່ໄດ້ຮັບຈາກ API ລົງໃນ state
10    } else {
11      message.error(response.message);
12    }
13    dispatch>HideLoading();
14  } catch (error) {
15    dispatch>HideLoading();
16    message.error(error.message);
17  }
18 };

```

```

1 // ພຶກຂັ້ນສ່າຫວັບລບໂຮງໝັ້ນ
2 const handleDelete = async (id) => {
3   try {
4     dispatch>ShowLoading();
5     const response = await DeleteTheatre({ theatreId: id });
6     // ເຮັດກໃຫ້ງານ API ເພື່ອລບໂຮງໝັ້ນໄດ້ໂດຍໃຫ້ ID ຂອງໂຮງໝັ້ນ
7     if (response.success) {
8       message.success(response.message);
9       getData(); // ໃນລດຂໍອມູລໂຮງໝັ້ນໃໝ່ກ່ຽວຂ້ອງລົງຈາກລບໂຮງໝັ້ນເສີ່ງສົມບຸຮະນີ
10    } else {
11      message.error(response.message);
12    }
13    dispatch>HideLoading();
14  } catch (error) {
15    dispatch>HideLoading();
16    message.error(error.message);
17  }
18 };

```

```
1 // គូលំនៅខែងតារាងរាយការទិន្នន័យ
2 const columns = [
3     {
4         title: "Name",
5         dataIndex: "name",
6     },
7     {
8         title: "Address",
9         dataIndex: "address",
10    },
11    {
12        title: "Phone",
13        dataIndex: "phone",
14    },
15    {
16        title: "Email",
17        dataIndex: "email",
18    },
19    {
20        title: "Status",
21        dataIndex: "isActive",
22        render: (text, record) => {
23            if (text) {
24                return "Active";
25            } else {
26                return "Pending";
27            }
28        },
29    },

```

```
1      {
2        title: "Action",
3        dataIndex: "action",
4        render: (text, record) => {
5          return (
6            <div className="flex gap-1 items-center">
7              <i
8                className="ri-delete-bin-line"
9                style={{ color: "red" }}
10               onClick={() => {
11                 handleDelete(record._id);
12               }}
13             ></i>
14             <i
15               className="ri-pencil-line"
16               style={{ color: "blue" }}
17               onClick={() => {
18                 setFormType("edit");
19                 setSelectedTheatre(record);
20                 setShowTheatreFormModal(true);
21               }}
22             ></i>
23             {record.isActive && (
24               <span
25                 className="underline"
26                 onClick={() => {
27                   setSelectedTheatre(record);
28                   setOpenShowsModal(true);
29                 }}
30               >
31                 Theatres Info
32               </span>
33             )}
34           </div>
35         );
36       },
37     },
38   ];
}
```

```

1 // តើងខំណួលទិន្នន័យអ៊ូគុមពិភេណនៃការទូទាត់ខ្លួនដោយបានរាយ
2 useEffect(() => {
3     getData(); // អ៊ូគុមពិភេណនៃការទូទាត់ នឹងតើងខំណួលទិន្នន័យ
4 }, []);
5
6 return (
7     <div>
8         {/* ផ្តល់ព័ត៌មានទិន្នន័យ */}
9         <div className="flex justify-end mb-1">
10             <Button
11                 variant="outlined"
12                 title="Add Theatre"
13                 onClick={() => {
14                     setFormType("add");
15                     setShowTheatreFormModal(true);
16                 }}
17             />
18         </div>
19
20         {/* គារចាយសែនាំខំណួលទិន្នន័យ */}
21         <Table columns={columns} dataSource={theatres} />
22
23         {/* ផែនការបង្កើតថ្មីនៃការទូទាត់ និងខំណួលទិន្នន័យ */}
24         {showTheatreFormModal && (
25             <TheatreForm
26                 showTheatreFormModal={showTheatreFormModal}
27                 setShowTheatreFormModal={setShowTheatreFormModal}
28                 formType={formType}
29                 setFormType={setFormType}
30                 selectedTheatre={selectedTheatre}
31                 setSelectedTheatre={setSelectedTheatre}
32                 getData={getData}
33             />
34         )}
35
36         {/* គារចាយសែនាំខំណួលទិន្នន័យ */}
37         {openShowsModal && (
38             <Shows
39                 openShowsModal={openShowsModal}
40                 setOpenShowsModal={setOpenShowsModal}
41                 theatre={selectedTheatre}
42             />
43         )}
44     </div>
45 );
46 }
47
48 export default TheatresList;

```

## โรงภาพยนตร์สำหรับรอบฉาย (Theatres for Movie)

### index.js

**คำอธิบาย :** ไฟล์นี้เป็นคอมโพเนนต์ที่แสดงรายชื่อโรงหนังที่มีการฉายหนังเรื่องนึงตามวันที่ที่กำหนด โดยแสดงข้อมูลหนังที่กำลังแสดง รวมถึงรายละเอียดของหนัง เช่น ชื่อ, ภาษา, ระยะเวลา, วันฉาย, และประเภท และแสดงรายชื่อโรงหนังที่มีการฉาย พร้อมกับเวลาที่ฉาย โดยสามารถคลิกที่เวลาที่ฉาย เพื่อไปยังหน้าจองตั๋วได้ ใช้งาน useEffect เพื่อดึงข้อมูลหนังและโรงหนังเมื่อคอมโพเนนต์โหลดเข้ามา และใช้งาน useParams เพื่อดึงค่า id ของหนังจาก URL และ useEffect เพื่อดึงข้อมูลโรงหนัง เมื่อมีการเปลี่ยนแปลงในวันที่ที่เลือก และแสดงข้อมูลโรงหนังในรูปแบบการ์ดที่มีรายชื่อหนังที่ฉาย



```

1  /**
2   * Component สำหรับแสดงรายชื่อโรงหนังที่มีการฉายหนังเรื่องนึงตามวันที่ที่กำหนด
3   * @returns {JSX.Element} คัด JSX สำหรับแสดงรายชื่อโรงหนังและรายละเอียดการฉายหนัง
4  */
5 function TheatresForMovie() {
6   // เก็บค่าวันที่ที่ถูกเลือกจาก query string หรือໃบ้ค่าปัจจุบัน
7   const tempDate = new URLSearchParams(window.location.search).get("date");
8   const [date, setDate] = React.useState(
9     tempDate || moment().format("YYYY-MM-DD")
10   );
11
12   const [movie, setMovie] = React.useState([]);
13   const [theatres, setTheatres] = React.useState([]);
14
15   const navigate = useNavigate();
16   const dispatch = useDispatch();
17   const params = useParams();

```



The screenshot shows a mobile application interface with a dark background. At the top, there are three circular icons: red, yellow, and green. Below them is a code editor window containing the following JavaScript code:

```
1  /**
2   * ดึงข้อมูลของหนังที่ค่าลั่งแสดงตาม id ที่ระบุใน URL params
3   */
4  const getData = async () => {
5    try {
6      dispatch>ShowLoading();
7      const response = await GetMovieById(params.id);
8      if (response.success) {
9        setMovie(response.data);
10     } else {
11       message.error(response.message);
12     }
13     dispatch>HideLoading();
14   } catch (error) {
15     dispatch>HideLoading();
16     message.error(error.message);
17   }
18 };
19
20 /**
21 * ดึงข้อมูลโรงหนังที่มีการฉายหนังเรื่องนึงตามวันที่ที่กำหนด
22 */
23 const getTheatres = async () => {
24   try {
25     dispatch>ShowLoading();
26     const response = await GetAllTheatresByMovie({ date, movie: params.id });
27     if (response.success) {
28       setTheatres(response.data);
29     } else {
30       message.error(response.message);
31     }
32     dispatch>HideLoading();
33   } catch (error) {
34     dispatch>HideLoading();
35     message.error(error.message);
36   }
37 };
38
39 useEffect(() => {
40   getData();
41 }, []);
42
43 useEffect(() => {
44   getTheatres();
45 }, [date]);
```

```
● ● ●  
1  return (  
2    movie && (  
3      <div>  
4        /* ພອມລົບຂອງໜັງ */  
5        <div className="flex justify-between items-center mb-2">  
6          <div>  
7            <h1 className="text-2xl uppercase">  
8              {movie.title} ({movie.language})  
9            </h1>  
10           <h1 className="text-md">Duration : {movie.duration} mins</h1>  
11           <h1 className="text-md">  
12             Release Date : {moment(movie.releaseDate).format("MMM Do yyyy")}  
13           </h1>  
14           <h1 className="text-md">Genre : {movie.genre}</h1>  
15         </div>  
16  
17         <div>  
18           <h1 className="text-md">Select Date</h1>  
19           <input  
20             type="date"  
21             min={moment().format("YYYY-MM-DD")}  
22             value={date}  
23             onChange={(e) => {  
24               setDate(e.target.value);  
25               navigate(`/movie/${params.id}?date=${e.target.value}`);  
26             }}  
27           />  
28         </div>  
29       </div>  
30     <hr />
```

```
1      /* รายชื่อโรงหนังที่มีการฉาย */
2  <div className="mt-1">
3      <h1 className="text-xl uppercase">Theatres</h1>
4  </div>
5
6  <div className="mt-1 flex flex-col gap-1">
7      {theatres.map((theatre) => (
8          <div className="card p-2">
9              <h1 className="text-md uppercase">{theatre.name}</h1>
10             <h1 className="text-sm">Address : {theatre.address}</h1>
11
12             <div className="divider"></div>
13
14             <div className="flex gap-2">
15                 {theatre.shows
16                     .sort(
17                         (a, b) => moment(a.time, "HH:mm") - moment(b.time, "HH:mm")
18                     )
19                     .map((show) => (
20                         <div
21                             className="card p-1 cursor-pointer"
22                             onClick={() => {
23                                 navigate(`book-show/${show._id}`);
24                             }}
25                         >
26                             <h1 className="text-sm">
27                                 {/* AM PM */}
28                                 {/* (moment(show.time, "HH:mm").format("hh:mm A")) */}
29
30                                 {/* 24 Hour */}
31                                 { (show.time) }
32                             </h1>
33                             <div>
34                                 ))
35                         </div>
36                     </div>
37                 )));
38             </div>
39         </div>
40     )
41 );
42 }
43
44 export default TheatresForMovie;
```

## การจัดการสถานะด้วย Redux (State Management with Redux)

### loadersSlice.js

คำอธิบาย : ไฟล์นี้เป็นส่วนของการสร้าง Slice ใน Redux Toolkit ที่ใช้สำหรับการจัดการกับสถานะของการโหลด โดยมีการสร้าง Slice ชื่อ loadersSlice ซึ่งมีสถานะเริ่มต้นคือ loading เป็นเท็จ และมี reducers สำหรับการตั้งค่า loading เป็นจริงและเท็จ

```

● ● ●

1 const loadersSlice = createSlice({
2   name: "loaders",
3   initialState: {
4     loading: false,
5   },
6   reducers: {
7     /**
8      * การกระทำเพื่อดึงค่าการโหลดเป็นจริง
9      * @function ShowLoading
10     * @memberof module:loadersSlice
11     * @param {LoadersState} state - อ้อมนใจค์สถานะปัจจุบัน
12     * @return {void}
13     */
14     ShowLoading: (state) => {
15       state.loading = true;
16     },
17     /**
18      * การกระทำเพื่อดึงค่าการโหลดเป็นเท็จ
19      * @function HideLoading
20      * @memberof module:loadersSlice
21      * @param {LoadersState} state - อ้อมนใจค์สถานะปัจจุบัน
22      * @return {void}
23      */
24     HideLoading: (state) => {
25       state.loading = false;
26     }
27   }
28 });
29
30 export const { ShowLoading, HideLoading } = loadersSlice.actions;
31 export default loadersSlice.reducer;

```

## store.js

คำอธิบาย : ไฟล์นี้เป็นส่วนของการสร้าง Redux store โดยใช้ configureStore จาก Redux Toolkit ซึ่งมีการกำหนด reducers สำหรับ loaders และ users



```
1  /**
2   * สร้าง store โดยใช้ configureStore จาก Redux Toolkit
3   * โดยกำหนด reducers สำหรับ loaders และ users
4   */
5 const store = configureStore({
6   reducer: {
7     loaders: loadersReducer,
8     users: usersReducer,
9   },
10 });
11
12 export default store;
```

## usersSlice.js

คำอธิบาย : ไฟล์นี้โค้ดด้านบนเป็นการใช้ createSlice จาก @reduxjs/toolkit เพื่อสร้าง slice สำหรับการจัดการข้อมูลผู้ใช้ โดยระบุชื่อ slice เป็น "users" และกำหนด initialState ให้กับ user เป็น null เพื่อรับว่าไม่มีผู้ใช้เริ่มต้น และมี reducer เดียวชื่อ "SetUser" ซึ่งใช้ในการกำหนดข้อมูลผู้ใช้ โดยรับ state และ action เพื่อกำหนดข้อมูลผู้ใช้ใหม่จาก payload ของ action และสุดท้าย export action creator SetUser และ reducer ของ slice นี้เพื่อใช้ในการสร้าง Redux store ได้.

```

1  /**
2   * สร้าง slice สำหรับการจัดการข้อมูลผู้ใช้
3   */
4  const usersSlice = createSlice({
5    name: "users", // ชื่อ slice
6    initialState: {
7      user: null, // สถานะเริ่มต้น: ไม่มีผู้ใช้
8    },
9    reducers: {
10      /**
11       * Reducer สำหรับการกำหนดข้อมูลผู้ใช้
12       * @param {Object} state -
13       * สถานะปัจจุบันของ slice
14       * @param {Object} action -
15       * ข้อมูลแอ็คชันที่ถูกเรียกใช้
16       */
17      SetUser: (state, action) => {
18        state.user = action.payload;
19        // กำหนดข้อมูลผู้ใช้จาก payload ที่ส่งมา
20      },
21    },
22  });
23
24  export const { SetUser } = usersSlice.actions;
25 // สร้าง action creator จาก reducers ที่กำหนด
26
27  export default usersSlice.reducer;
28 // ส่งออก reducer ของ slice นี้

```

## การกำหนดค่าเซิร์ฟเวอร์ (Server Configuration)

### dbConfig.js

คำอธิบาย : ไฟล์นี้เป็นการใช้ Mongoose เพื่อเชื่อมต่อกับฐานข้อมูล MongoDB โดยการตั้งค่าคิวรี่โดยเคร่งความเข้มงวดเป็นเท็จ และเชื่อมต่อกับ MongoDB โดยใช้ URL ที่ระบุในตัวแปรสิงแผลล้อม mongo\_url และคืนค่าการเชื่อมต่อกับฐานข้อมูล MongoDB ผ่าน mongoose.Connection โดยมีการตรวจสอบเหตุการณ์การเชื่อมต่อเพื่อแสดงข้อความที่เกี่ยวข้องบนコンโซล.



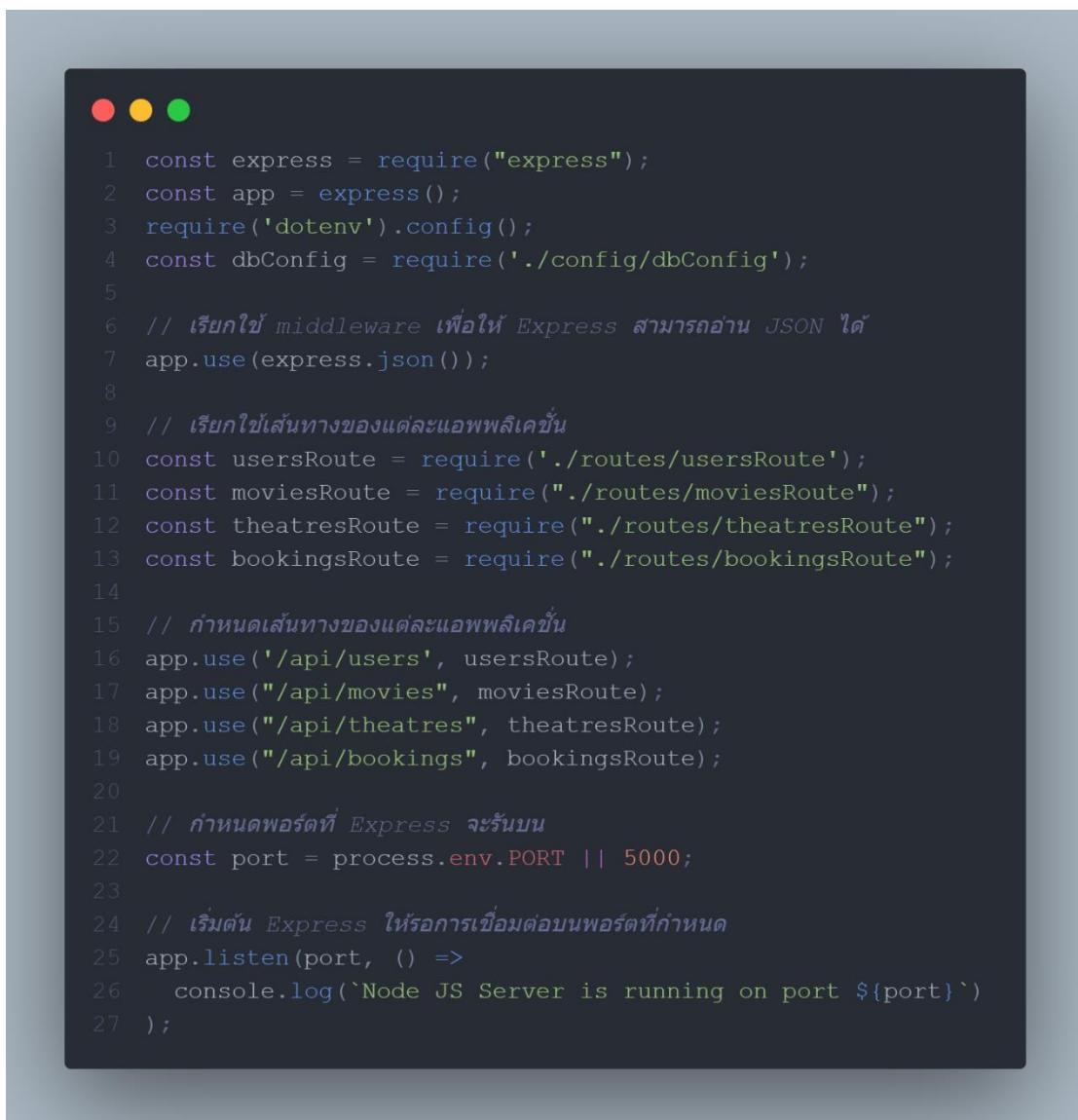
```

1 const mongoose = require('mongoose');
2
3 /**
4  * ตั้งค่าการใช้งานคิวรี่โดยเคร่งความเข้มงวดเป็นเท็จใน Mongoose.
5  */
6 mongoose.set('strictQuery', false)
7
8 /**
9  * เชื่อมต่อกับฐานข้อมูล MongoDB โดยใช้ URL ที่กำหนดในตัวแปรสิงแผลล้อม mongo_url.
10 * @param {string} process.env.mongo_url URL ของ MongoDB
11 * @return {mongoose.Connection} การเชื่อมต่อกับฐานข้อมูล MongoDB
12 */
13 mongoose.connect(process.env.mongo_url)
14
15 const connection = mongoose.connection;
16
17 connection.on('connected' , ()=>{
18     console.log('Mongo DB Connected');
19 })
20
21 connection.on('error' , (err)=>{
22     console.log('Mongo DB Failed');
23 })

```

## server.js

**คำอธิบาย :** Express server ถูกสร้างขึ้นเพื่อรับและตอบคำขอ โดยมีการใช้ middleware เพื่อให้ Express สามารถอ่าน JSON และกำหนดเส้นทางสำหรับแต่ละแอปพลิเคชัน เมื่อมีการเข้ามายัง Express จะทำงานบนพอร์ตที่กำหนด และแสดงข้อความเพื่อแจ้งให้ทราบว่าเซิร์ฟเวอร์กำลังทำงานอยู่ที่พอร์ตที่กำหนดไว้ โดยมีการเรียกใช้เส้นทางของแต่ละแอปพลิเคชัน เช่น usersRoute, moviesRoute, theatresRoute, และ bookingsRoute เพื่อรับคำขอต่าง ๆ ที่เข้ามาผ่านพอร์ตที่กำหนดไว้ในตัวแปร app ของ Express โดยทำให้ Express สามารถตอบคำขอจากแต่ละเส้นทางนั้นๆ ได้ในลำดับที่เหมาะสม



```

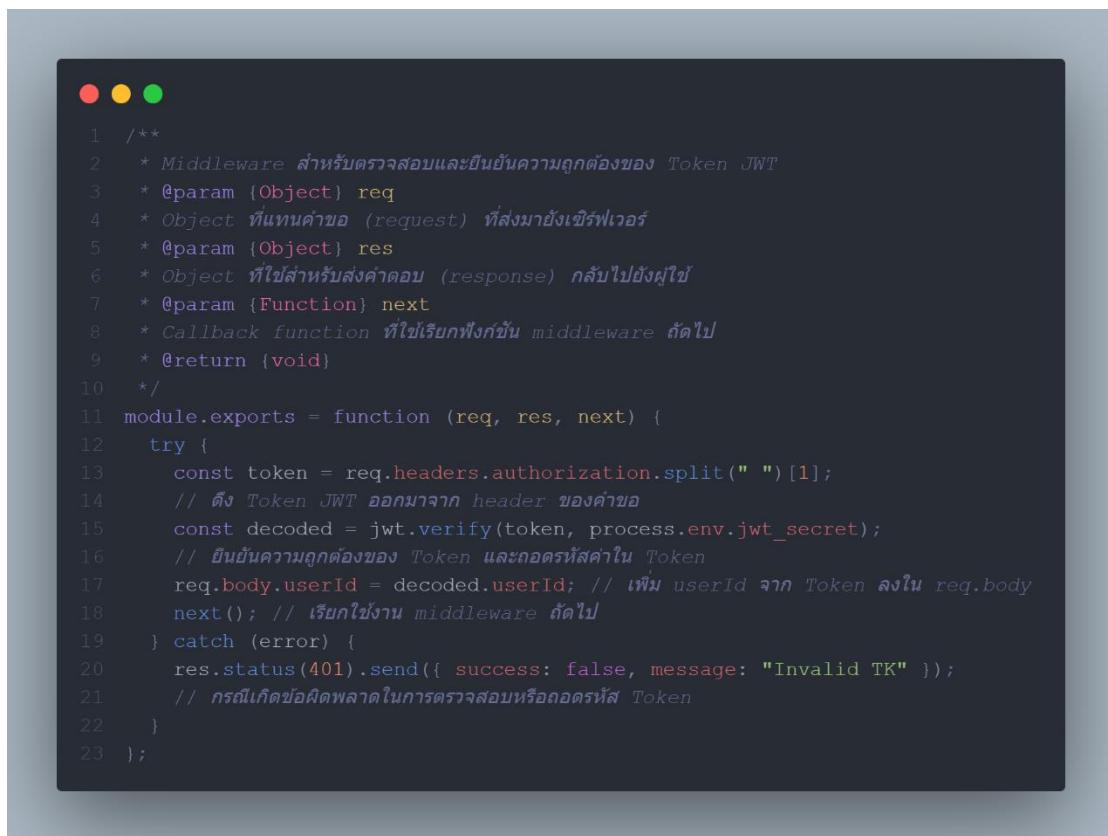
1 const express = require("express");
2 const app = express();
3 require('dotenv').config();
4 const dbConfig = require('./config/dbConfig');
5
6 // เรียกใช้ middleware เพื่อให้ Express สามารถอ่าน JSON ได้
7 app.use(express.json());
8
9 // เรียกใช้เส้นทางของแต่ละแอปพลิเคชัน
10 const usersRoute = require('./routes/usersRoute');
11 const moviesRoute = require("./routes/moviesRoute");
12 const theatresRoute = require("./routes/theatresRoute");
13 const bookingsRoute = require("./routes/bookingsRoute");
14
15 // กำหนดเส้นทางของแต่ละแอปพลิเคชัน
16 app.use('/api/users', usersRoute);
17 app.use("/api/movies", moviesRoute);
18 app.use("/api/theatres", theatresRoute);
19 app.use("/api/bookings", bookingsRoute);
20
21 // กำหนดพอร์ตที่ Express จะรันบน
22 const port = process.env.PORT || 5000;
23
24 // เริ่มต้น Express ให้รับคำขอบนพอร์ตที่กำหนด
25 app.listen(port, () =>
26   console.log(`Node JS Server is running on port ${port}`)
27 );

```

## มิดเดิลแวร์ (Middlewares)

### authMiddleware.js

**คำอธิบาย :** ใช้เพื่อตรวจสอบความถูกต้องของ Token JWT ที่ส่งมา กับคำขอ โดยมีขั้นตอนดังนี้ ดึง Token JWT จาก header ของคำขอ จากนั้นทำการยืนยันความถูกต้องของ Token และถอดรหัสข้อมูลภายใน Token หลังจากนั้น เพิ่ม userId จาก Token ลงใน req.body และทำการเรียกใช้ middleware ถัดไปในกรณีที่ไม่พบข้อผิดพลาด ในกรณีที่มีข้อผิดพลาดในการตรวจสอบหรือถอดรหัส Token จะส่งคำตอบกลับด้วยสถานะ 401 และข้อความว่า "Invalid Token" แสดงว่าการตรวจสอบไม่ผ่านการตรวจสอบหรือถอดรหัส Token ล้มเหลว



```

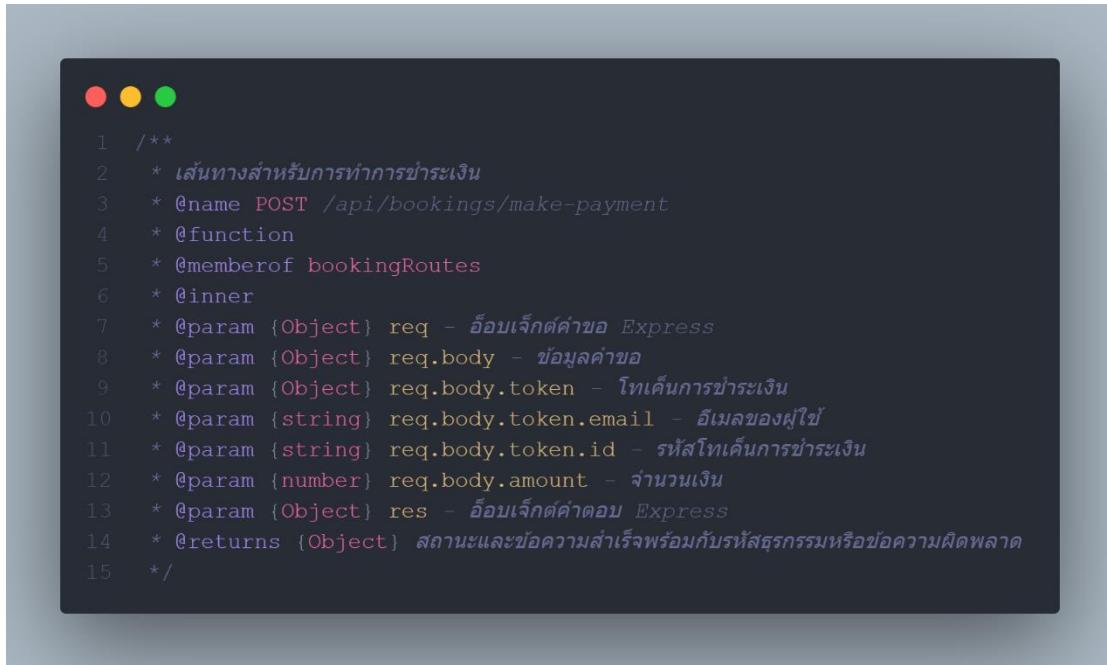
1  /**
2   * Middleware สำหรับตรวจสอบและยืนยันความถูกต้องของ Token JWT
3   * @param {Object} req ที่ส่งมาอย่างเช่นฟอร์ม
4   * @param {Object} res ที่ใช้สำหรับส่งค่าตอบ
5   * @param {Function} next Callback function ที่ให้รีบก็องก์น middleware ถัดไป
6   * @return {void}
7  */
8 module.exports = function (req, res, next) {
9   try {
10     const token = req.headers.authorization.split(" ")[1];
11     // ดึง Token JWT ออกมาจาก header ของคำขอ
12     const decoded = jwt.verify(token, process.env.jwt_secret);
13     // ยืนยันความถูกต้องของ Token และถอดรหัสค่าใน Token
14     req.body.userId = decoded.userId; // เพิ่ม userId จาก Token ลงใน req.body
15     next(); // เรียกใช้งาน middleware ถัดไป
16   } catch (error) {
17     res.status(401).send({ success: false, message: "Invalid TK" });
18     // กรณีเกิดข้อผิดพลาดในการตรวจสอบหรือถอดรหัส Token
19   }
20 }
21
22
23

```

## เส้นทางเซิร์ฟเวอร์ (Server Routes)

### bookingsRoute.js

**คำอธิบาย :** เส้นทางการทำการทำชำระเงินใน Express ใช้ middleware เพื่อตรวจสอบการยืนยันตัวตนก่อนชำระเงิน โดยจะสร้างลูกค้าใหม่ใน Stripe และทำการชำระเงิน หลังจากนั้นจะส่งข้อความสถานะและรหัสธุรกรรมกลับไปยังผู้ใช้ การจองตัวการแสดงใช้ middleware เพื่อตรวจสอบการยืนยันตัวตนก่อนจอง หลังจากทำการจองตัว Express จะบันทึกข้อมูลการจองและอัปเดตที่นั่งที่ถูกจอง และส่งข้อความสถานะและข้อมูลการจองกลับไปยังผู้ใช้ สำหรับการดึงข้อมูลการจองทั้งหมดของผู้ใช้ Express จะค้นหาและดึงข้อมูลการจองจากฐานข้อมูล และส่งข้อความสถานะพร้อมกับข้อมูลการจองกลับไปยังผู้ใช้



```

1  /**
2   * เส้นทางสำหรับการทำการทำชำระเงิน
3   * @name POST /api/bookings/make-payment
4   * @function
5   * @memberof bookingRoutes
6   * @inner
7   * @param {Object} req - อ็อบเจ็กต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {Object} req.body.token - โทเค็นการทำชำระเงิน
10  * @param {string} req.body.token.email - อีเมลของผู้ใช้
11  * @param {string} req.body.token.id - รหัสโทเค็นการทำชำระเงิน
12  * @param {number} req.body.amount - จำนวนเงิน
13  * @param {Object} res - อ็อบเจ็กต์ค่าตอบ Express
14  * @returns {Object} สถานะและข้อความสำเร็จพร้อมกับรหัสธุรกรรมหรือข้อความผิดพลาด
15 */

```

```
1 router.post("/make-payment", authMiddleware, async (req, res) => {
2   try {
3     const { token, amount } = req.body;
4
5     const customer = await stripe.customers.create({
6       email: token.email,
7       source: token.id,
8     });
9
10    const charge = await stripe.charges.create({
11      amount,
12      currency: "thb",
13      customer: customer.id,
14      receipt_email: token.email,
15      description: "Ticket Booked for Movie",
16    });
17
18    const transactionId = charge.id;
19
20    res.send({
21      success: true,
22      message: "Payment successful",
23      data: transactionId,
24    });
25  } catch (error) {
26    res.send({
27      success: false,
28      message: error.message,
29    });
30  }
31});
```

```
1  /**
2   * เส้นทางสำหรับการจองตั๋วการแสดง
3   * @name POST /api/bookings/book-show
4   * @function
5   * @memberof bookingRoutes
6   * @inner
7   * @param {Object} req - อ้อมเบล็กด์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {Object} req.body.show - รายละเอียดการแสดง
10  * @param {string} req.body.show._id - รหัสการแสดง
11  * @param {string[]} req.body.seats - อาร์เรย์ของหมายเลขที่นั่ง
12  * @param {Object} res - อ้อมเบล็กด์ค่าตอบ Express
13  * @returns {Object} สถานะและข้อความสำเร็จพร้อมกับข้อมูลการจองใหม่หรือข้อความผิดพลาด
14 */
15 router.post("/book-show", authMiddleware, async (req, res) => {
16   try {
17     // บันทึกการจอง
18     const newBooking = new Booking(req.body);
19     await newBooking.save();
20
21     const show = await Show.findById(req.body.show);
22     // อัปเดตที่นั่ง
23     await Show.findByIdAndUpdate(req.body.show, {
24       bookedSeats: [...show.bookedSeats, ...req.body.seats],
25     });
26
27     res.send({
28       success: true,
29       message: "Show booked successfully",
30       data: newBooking,
31     });
32   } catch (error) {
33     res.send({
34       success: false,
35       message: error.message,
36     });
37   }
38 });


```

```
1  /**
2   * เส้นทางสำหรับการดึงข้อมูลการจองห้องหนังของผู้ใช้
3   * @name GET /api/bookings/get-bookings
4   * @function
5   * @memberof bookingRoutes
6   * @inner
7   * @param {Object} req - อีองเจ็กต์ค่าของ Express
8   * @param {string} req.body.userId - รหัสผู้ใช้
9   * @param {Object} res - อีองเจ็กต์ค่าตอบ Express
10  * @returns {Object} สถานะและข้อความสำหรับการรับรองว่าการจองหนังสำเร็จพร้อมกับการเรียกใช้งาน
11 */
12 router.get("/get-bookings", authMiddleware, async (req, res) => {
13   try {
14     const bookings = await Booking.find({ user: req.body.userId })
15       .populate("show")
16       .populate({
17         path: "show",
18         populate: {
19           path: "movie",
20           model: "movies",
21         },
22       })
23       .populate("user")
24       .populate({
25         path: "show",
26         populate: {
27           path: "theatre",
28           model: "theatres",
29         },
30       });
31
32     res.send({
33       success: true,
34       message: "Bookings fetched successfully",
35       data: bookings,
36     });
37   } catch (error) {
38     res.send({
39       success: false,
40       message: error.message,
41     });
42   }
43 });
44
45 module.exports = router;
```

## moviesRoute.js

**คำอธิบาย :** เส้นทางการจัดการข้อมูลภาพยนตร์ใน Express ประกอบด้วยการเพิ่มภาพยนตร์ใหม่ การแสดงภาพยนตร์ทั้งหมด การอัปเดตภาพยนตร์ และการลบภาพยนตร์ โดยใช้ middleware เพื่อตรวจสอบการยืนยันตัวตนก่อนดำเนินการทุกรอบ และมีเส้นทางเพิ่มเติมสำหรับดึงข้อมูลภาพยนตร์ตามรหัสที่ระบุ การจัดการข้อมูลจะส่งค่าตอบกลับในรูปแบบข้อความสถานะพร้อมกับข้อมูลหรือข้อความผิดพลาดตามบรรทัดที่ทำงานแต่ละครั้ง



```

1  /**
2   * เส้นทางสำหรับเพิ่มภาพยนตร์ใหม่
3   * @name POST /api/movies/add-movie
4   * @function
5   * @memberof movieRoutes
6   * @inner
7   * @param {Object} req - อ็อบเจกต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลภาพยนตร์ที่จะเพิ่ม
9   * @param {Object} res - อ็อบเจกต์ค่าตอบ Express
10  * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
11 */
12 router.post("/add-movie", authMiddleware, async (req, res) => {
13   try {
14     const newMovie = new Movie(req.body);
15     await newMovie.save();
16     res.send({
17       success: true,
18       message: "Movie now added",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });

```



```
1  /**
2   * เส้นทางสำหรับแสดงภาพยนตร์ทั้งหมด
3   * @name GET /api/movies/get-all-movies
4   * @function
5   * @memberof movieRoutes
6   * @inner
7   * @param {Object} req - อี对象ค่าของ Express
8   * @param {Object} res - อีobjectค่าตอบของ Express
9   * @returns {Object} สถานะและข้อมูลความสำเร็จพร้อมกับอาร์เรย์ของภาพยนตร์หรือข้อความผิดพลาด
10  */
11 router.get("/get-all-movies", async (req, res) => {
12   try {
13     const movies = await Movie.find().sort({ createdAt: -1 });
14     res.send({
15       success: true,
16       message: "Movies fetched successfully",
17       data: movies,
18     });
19   } catch (error) {
20     res.send({
21       success: false,
22       message: error.message,
23     });
24   }
25 });
```



```
1  /**
2   * เส้นทางสำหรับอัปเดตภาพยนตร์
3   * @name POST /api/movies/update-movie
4   * @function
5   * @memberof movieRoutes
6   * @inner
7   * @param {Object} req - อี-objectค่าของ Express
8   * @param {Object} req.body - ข้อมูลที่จะใช้ในการอัปเดตภาพยนตร์
9   * @param {string} req.body.movieId - รหัสภาพยนตร์ที่ต้องการอัปเดต
10  * @param {Object} res - อี-objectค่าตอบ Express
11  * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
12 */
13 router.post("/update-movie", authMiddleware, async (req, res) => {
14   try {
15     await Movie.findByIdAndUpdate(req.body.movieId, req.body);
16     res.send({
17       success: true,
18       message: "Movie updated",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });
```



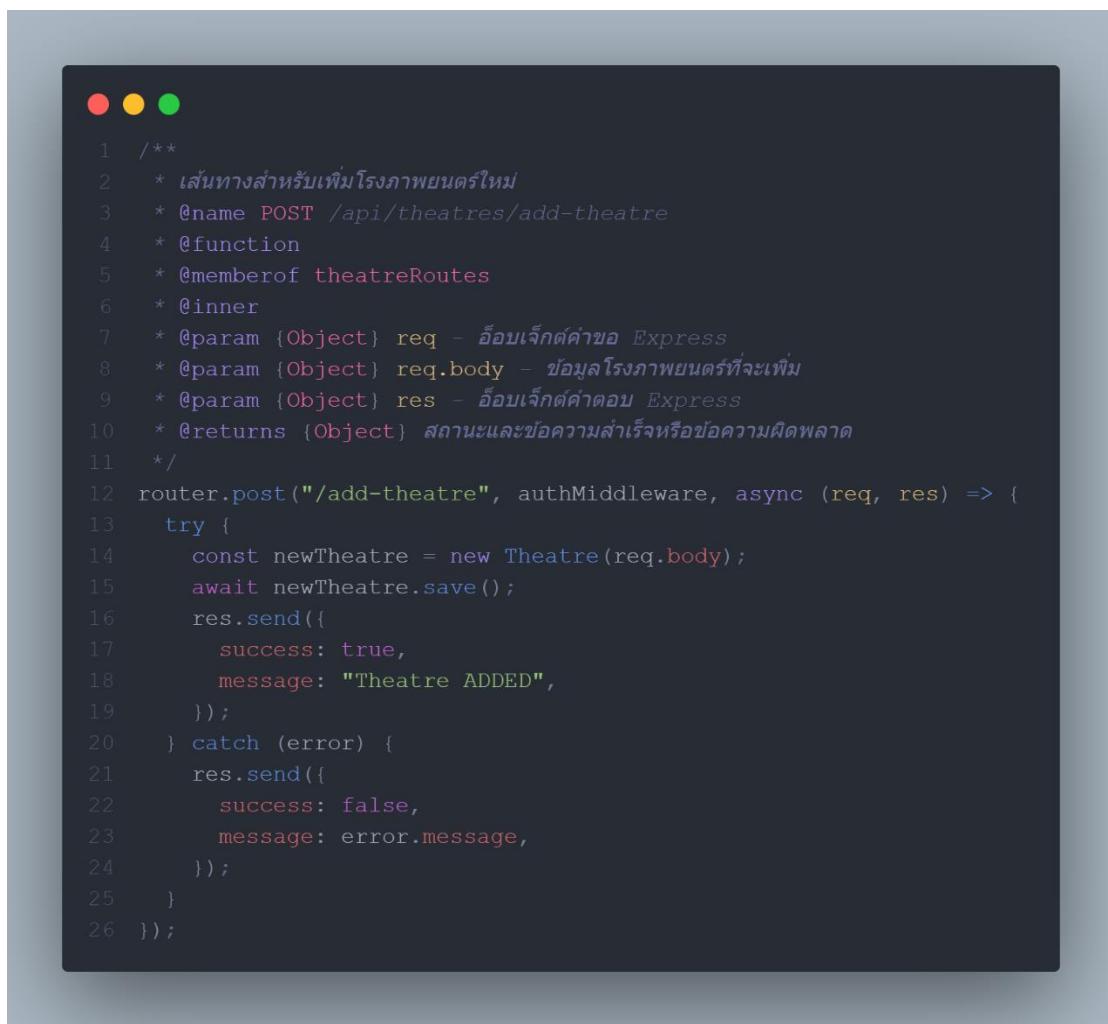
```
1  /**
2   * เส้นทางสำหรับลบภาพยนตร์
3   * @name POST /api/movies/delete-movie
4   * @function
5   * @memberof movieRoutes
6   * @inner
7   * @param {Object} req - อีอเบเจ็กต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.movieId - รหัสภาพยนตร์ที่ต้องการลบ
10  * @param {Object} res - อีอเบเจ็กต์ค่าตอบ Express
11  * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
12 */
13 router.post("/delete-movie", authMiddleware, async (req, res) => {
14   try {
15     await Movie.findByIdAndDelete(req.body.movieId);
16     res.send({
17       success: true,
18       message: "Movie deleted",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });
```



```
1  /**
2   * เส้นทางสำหรับดึงข้อมูลภาพยนตร์ตามรหัส
3   * @name GET /api/movies/get-movie-by-id/:id
4   * @function
5   * @memberof movieRoutes
6   * @inner
7   * @param {Object} req - อีโอบเจกต์ค่าของ Express
8   * @param {string} req.params.id - รหัสภาพยนตร์
9   * @param {Object} res - อีโอบเจกต์ค่าตอบ Express
10  * @returns {Object} สถานะและข้อมูลภาพยนตร์หรือข้อความผิดพลาด
11 */
12 router.get("/get-movie-by-id/:id", async (req, res) => {
13   try {
14     const movie = await Movie.findById(req.params.id);
15     res.send({
16       success: true,
17       message: "Movie fetched",
18       data: movie,
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });
27
28 module.exports = router;
```

## theatresRoute.js

**คำอธิบาย :** เส้นทางการจัดการข้อมูลโรงภาพยนตร์ใน Express รวมถึงการเพิ่มโรงภาพยนตร์ใหม่ ดึงข้อมูลโรงภาพยนตร์ทั้งหมด ดึงข้อมูลโรงภาพยนตร์โดยเจ้าของ อัปเดตข้อมูลโรงภาพยนตร์ ลบข้อมูลโรงภาพยนตร์ เพิ่มการแสดงภาพยนตร์ใหม่ ดึงข้อมูลการแสดงทั้งหมดโดยใช้โรงภาพยนตร์ ลบทึ่งการแสดงภาพยนตร์ในโรงภาพยนตร์ ดึงโรงภาพยนตร์ที่ไม่ซ้ำกันทั้งหมดสำหรับภาพยนตร์และวันที่ที่กำหนด และดึงข้อมูลการแสดงตามรหัสการแสดงที่กำหนด โดยมีการใช้ middleware เพื่อตรวจสอบการยืนยันตัวตนก่อนดำเนินการทุกรอบ และการจัดการข้อมูลจะส่งค่าตอบกลับในรูปแบบข้อความสถานะพร้อมกับข้อมูลหรือข้อความผิดพลาดตามบรรทัดที่ทำงานแต่ละครั้ง



```

1  /**
2   * เส้นทางสำหรับเพิ่มโรงภาพยนตร์ใหม่
3   * @name POST /api/theatres/add-theatre
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - ชื่อ昂เจ็กต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลโรงภาพยนตร์ที่จะเพิ่ม
9   * @param {Object} res - ชื่อ昂เจ็กต์ค่าตอบของ Express
10  * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
11 */
12 router.post("/add-theatre", authMiddleware, async (req, res) => {
13   try {
14     const newTheatre = new Theatre(req.body);
15     await newTheatre.save();
16     res.send({
17       success: true,
18       message: "Theatre ADDED",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });

```



```
1  /**
2   * เลันทางส่าหรับดึงข้อมูลโรงภาพยนตร์ทั้งหมด
3   * @name GET /api/theatres/get-all-theatres
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อีโอบเจ็กต์ค่าของ Express
8   * @param {Object} res - อีโอบเจ็กต์ค่าตอบ Express
9   * @returns {Object} สถานะและข้อมูลความสำเร็จพร้อมกับอาร์เรย์ของโรงภาพยนตร์หรือข้อมูลผิดพลาด
10 */
11 router.get("/get-all-theatres", authMiddleware, async (req, res) => {
12   try {
13     const theatres = await Theatre.find()
14       .populate("owner")
15       .sort({ createdAt: -1 });
16     res.send({
17       success: true,
18       message: "Theatres fetched",
19       data: theatres,
20     });
21   } catch (error) {
22     res.send({
23       success: false,
24       message: error.message,
25     });
26   }
27 });
```

```
● ● ●  
1  /**
2   * เส้นทางสำหรับดึงข้อมูลโรงภาพยนตร์โดยเจ้าของ
3   * @name POST /api/theatres/get-all-theatres-by-owner
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อ็อบเจกต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.owner - รหัสเจ้าของโรงภาพยนตร์
10  * @param {Object} res - อ็อบเจกต์ค่าตอบ Express
11  * @returns {Object} สถานะและข้อมูลความสำเร็จพร้อมกับรายการโรงภาพยนตร์ที่รีบอัปเดตตามผู้ดูแล
12 */
13 router.post("/get-all-theatres-by-owner", authMiddleware, async (req, res) => {
14   try {
15     const theatres = await Theatre.find({ owner: req.body.owner }).sort({
16       createdAt: -1,
17     });
18     res.send({
19       success: true,
20       message: "Theatres fetched by owner",
21       data: theatres,
22     });
23   } catch (error) {
24     res.send({
25       success: false,
26       message: error.message,
27     });
28   }
29 });
```



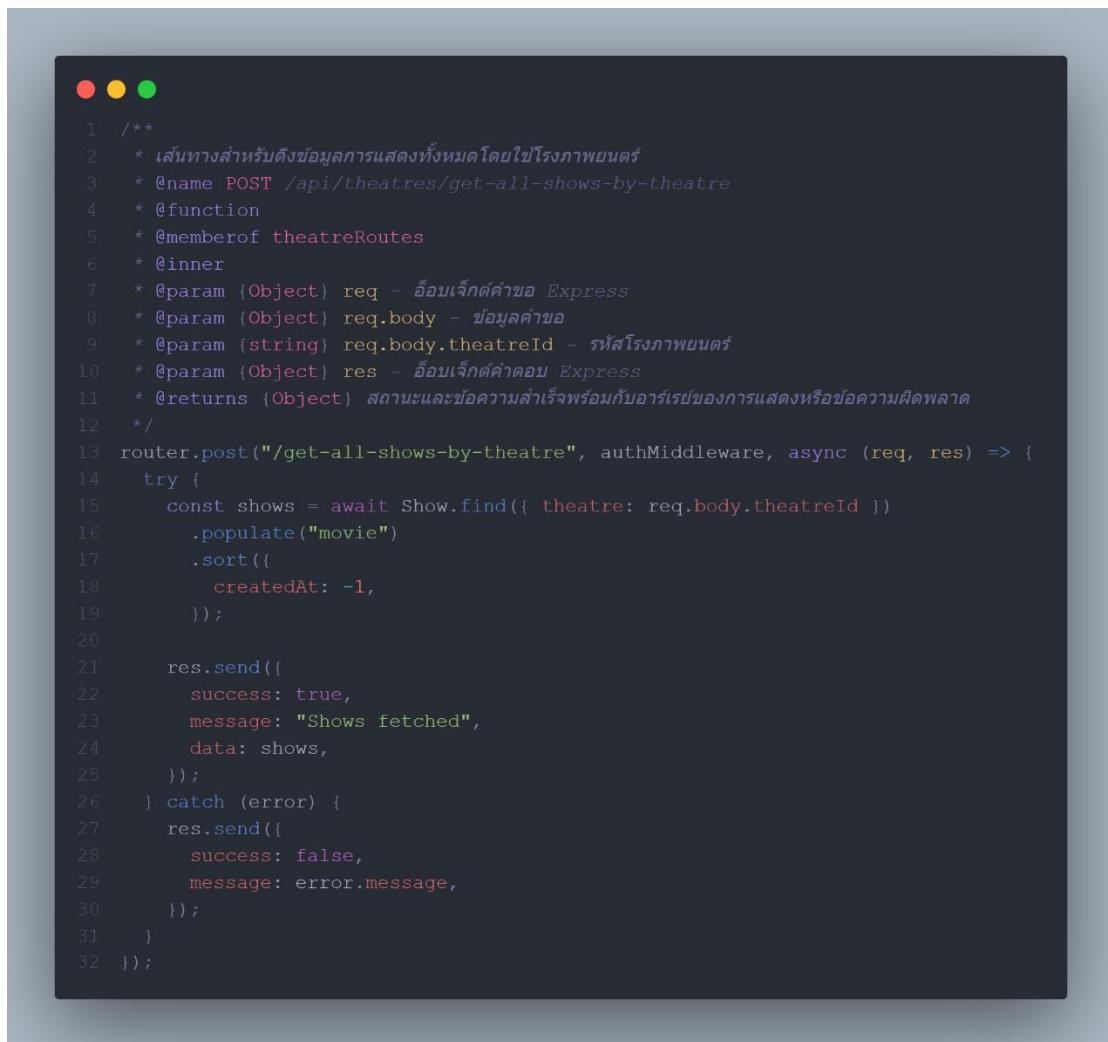
```
1  /**
2   * เล่นทางสำหรับอัปเดตข้อมูลโรงภาพยนตร์
3   * @name POST /api/theatres/update-theatre
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อีองเจกต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.theatreId - รหัสโรงภาพยนตร์ที่จะอัปเดต
10  * @param {Object} req.body - ข้อมูลที่ต้องการอัปเดต
11  * @param {Object} res - อีองเจกต์ค่าตอบ Express
12  * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
13 */
14 router.post("/update-theatre", authMiddleware, async (req, res) => {
15   try {
16     await Theatre.findByIdAndUpdate(req.body.theatreId, req.body);
17     res.send({
18       success: true,
19       message: "Theatre updated",
20     });
21   } catch (error) {
22     res.send({
23       success: false,
24       message: error.message,
25     });
26   }
27 });
```

```
1  /**
2   * เล่นทางสำหรับลบข้อมูลโรงภาพยนตร์
3   * @name POST /api/theatres/delete-theatre
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อ็อกเจ็กต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.theatreId - รหัสโรงภาพยนตร์ที่จะลบ
10  * @param {Object} res - อ็อกเจ็กต์ค่าตอบ Express
11  * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
12 */
13 router.post("/delete-theatre", authMiddleware, async (req, res) => {
14   try {
15     await Theatre.findByIdAndDelete(req.body.theatreId);
16     res.send({
17       success: true,
18       message: "Theatre deleted",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });


```

```
1  /**
2  * เส้นทางสำหรับเพิ่มการแสดงภาพยนตร์ใหม่
3  * @name POST /api/theatres/add-show
4  * @function
5  * @memberof theatreRoutes
6  * @inner
7  * @param {Object} req - อ็อบเจกต์ค่าของ Express
8  * @param {Object} req.body - ข้อมูลค่าของ
9  * @param {Object} res - อ็อบเจกต์ค่าตอบ Express
10 * @returns {Object} สถานะและข้อความสำเร็จหรือข้อความผิดพลาด
11 */
12 router.post("/add-show", authMiddleware, async (req, res) => {
13   try {
14     const newShow = new Show(req.body);
15     await newShow.save();
16     res.send({
17       success: true,
18       message: "Show added",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });


```



```
1  /**
2   * เล่นทางสำหรับดึงข้อมูลการแสดงทั้งหมดโดยใช้โรงภาพยนตร์
3   * @name POST /api/theatres/get-all-shows-by-theatre
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อีคอมเจ็กต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.theatreId - รหัสโรงภาพยนตร์
10  * @param {Object} res - อีคอมเจ็กต์ค่าตอบ Express
11  * @returns {Object} สถานะและข้อมูลการแสดงสำหรับร้องกับการเรียกใช้งานแสดงหรือข้อมูลความคิดเห็น
12 */
13 router.post("/get-all-shows-by-theatre", authMiddleware, async (req, res) => {
14   try {
15     const shows = await Show.find({ theatre: req.body.theatreId })
16       .populate("movie")
17       .sort({
18         createdAt: -1,
19       });
20
21     res.send({
22       success: true,
23       message: "Shows fetched",
24       data: shows,
25     });
26   } catch (error) {
27     res.send({
28       success: false,
29       message: error.message,
30     });
31   }
32 });


```



```
1  /**
2   * เส้นทางสำหรับการแสดงภาพยนตร์ในโรงภาพยนตร์
3   * @name POST /api/theatres/delete-show
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อ็อบเจกต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.showId - รหัสการแสดงภาพยนตร์ที่จะลบ
10  * @param {Object} res - อ็อบเจกต์ค่าตอบ Express
11  * @returns {Object} สถานะและข้อความสำหรับการลบภาพยนตร์
12 */
13 router.post("/delete-show", authMiddleware, async (req, res) => {
14   try {
15     await Show.findByIdAndDelete(req.body.showId);
16     res.send({
17       success: true,
18       message: "Show deleted",
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });
```

```

1  /**
2   * เส้นทางล่าหาห้องดูหนังที่ไม่มีข้ากันห้องหนังดูหนังตามดูหนังตามห้องหนังดูหนังตามห้องหนังตามห้องหนัง
3   * @name POST /api/theatres/get-all-theatres-by-movie
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อ้อมเจกต์คานอ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.movie - รหัสภาพยนตร์
10  * @param {string} req.body.date - วันที่
11  * @param {Object} res - อ้อมเจกต์คานอ Express
12  * @returns {Object} สถานะและข้อมูลความล่าเร็วของโรงภาพยนตร์หรือข้อมูลความผิดพลาด
13 */
14 router.post("/get-all-theatres-by-movie", authMiddleware, async (req, res) => {
15   try {
16     const { movie, date } = req.body;
17
18     // การแสดงภาพยนตร์
19     const shows = await Show.find({ movie, date })
20       .populate("theatre")
21       .sort({ createdAt: -1 });
22
23     // โรงภาพยนตร์ที่ไม่มีข้ากัน
24     let uniqueTheatres = [];
25     shows.forEach((show) => {
26       const theatre = uniqueTheatres.find(
27         (theatre) => theatre._id === show.theatre._id
28       );
29
30       if (!theatre) {
31         const showsForThisTheatre = shows.filter(
32           (showObj) => showObj.theatre._id === show.theatre._id
33         );
34         uniqueTheatres.push({
35           ...show.theatre._doc,
36           shows: showsForThisTheatre,
37         });
38       }
39     });
40
41     res.send({
42       success: true,
43       message: "Theatres fetched successfully",
44       data: uniqueTheatres,
45     });
46   } catch (error) {
47     res.send({
48       success: false,
49       message: error.message,
50     });
51   }
52 });

```



```
1  /**
2   * เส้นทางสำหรับดึงข้อมูลการแสดงตามรหัสการแสดงที่กำหนด
3   * @name POST /api/theatres/get-show-by-id
4   * @function
5   * @memberof theatreRoutes
6   * @inner
7   * @param {Object} req - อ็อบเจกต์ค่าของ Express
8   * @param {Object} req.body - ข้อมูลค่าของ
9   * @param {string} req.body.showId - รหัสการแสดง
10  * @param {Object} res - อ็อบเจกต์ค่าตอบ Express
11  * @returns {Object} สถานะและข้อมูลสำหรับรับกับข้อมูลการแสดงหรือข้อมูลผิดพลาด
12 */
13 router.post("/get-show-by-id", authMiddleware, async (req, res) => {
14   try {
15     const show = await Show.findById(req.body.showId)
16       .populate("movie")
17       .populate("theatre");
18     res.send({
19       success: true,
20       message: "Show fetched",
21       data: show,
22     });
23   } catch (error) {
24     res.send({
25       success: false,
26       message: error.message,
27     });
28   }
29 });
30
31 module.exports = router;
```

## usersRoute.js

**คำอธิบาย :** เส้นทางผู้ใช้: ลงทะเบียน, เข้าสู่ระบบ, และดึงข้อมูลผู้ใช้ปัจจุบันโดยมีการใช้ middleware สำหรับการตรวจสอบ Token ที่สร้างจากการเข้าสู่ระบบ เพื่อความปลอดภัยและการยืนยันตัวตน ในลงทะเบียนจะตรวจสอบอีเมลถูกใช้งานแล้วหรือไม่และแฮชรหัสผ่านก่อนบันทึกข้อมูล สำหรับการเข้าสู่ระบบ จะตรวจสอบความถูกต้องของข้อมูลและสร้าง Token เพื่อการยืนยันตัวตน สุดท้าย การดึงข้อมูลผู้ใช้จะส่งข้อมูลโดยไม่รวมรหัสผ่าน

```

1  /**
2  * เส้นทางสำหรับลงทะเบียนผู้ใช้ใหม่
3  * @name POST /api/users/register
4  * @function
5  * @memberof userRoutes
6  * @inner
7  * @param {Object} req - อีคอมเจ็กต์ค่าของ Express
8  * @param {Object} req.body - ข้อมูลค่าของ
9  * @param {string} req.body.email - อีเมลของผู้ใช้
10 * @param {string} req.body.password - รหัสผ่านของผู้ใช้
11 * @param {Object} res - อีคอมเจ็กต์ค่าตอบ Express
12 * @returns {Object} สถานะและความสำเร็จหรือข้อมูลผิดพลาด
13 */
14 router.post("/register", async (req, res) => {
15   try {
16     // ตรวจสอบอีเมลที่ไม่ซ้ำกัน
17     const usersExists = await User.findOne({ email: req.body.email });
18     if (usersExists) {
19       return res.send({
20         success: false,
21         message: "This user already exists.",
22       });
23     }
24
25     // แฮชรหัสผ่าน
26     const salt = await bcrypt.genSalt(10);
27     const hashedPassword = await bcrypt.hash(req.body.password, salt);
28     req.body.password = hashedPassword;
29
30     // บันทึกข้อมูลผู้ใช้
31     const newUser = new User(req.body);
32     await newUser.save();
33
34     res.send({ success: true, message: "Successfully created a user account!" });
35   } catch (error) {
36     res.send({
37       success: false,
38       message: error.message,
39     });
40   }
41 });

```

```

1  /**
2  * เส้นทางสำหรับเข้าสู่ระบบ
3  * @name POST /api/users/login
4  * @function
5  * @memberof userRoutes
6  * @inner
7  * @param {Object} req - อีองบเจ็กต์ค่าของ Express
8  * @param {Object} req.body - ข้อมูลค่าของ
9  * @param {string} req.body.email - อีเมลของผู้ใช้
10 * @param {string} req.body.password - รหัสผ่านของผู้ใช้
11 * @param {Object} res - อีองบเจ็กต์ค่าตอบ Express
12 * @returns {Object} สถานะและข้อความสำเร็จพร้อมกับ Token หรือข้อความผิดพลาด
13 */
14 router.post("/login", async (req, res) => {
15   try {
16     // ตรวจสอบการเข้าสู่ระบบ
17     const user = await User.findOne({ email: req.body.email });
18     if (!user) {
19       return res.send({
20         success: false,
21         message: "Incorrect email or password!",
22       });
23     }
24
25     // ตรวจสอบรหัสผ่าน
26     const validPassword = await bcrypt.compare(
27       req.body.password,
28       user.password
29     );
30
31     if (!validPassword) {
32       return res.send({
33         success: false,
34         message: "Incorrect email or password!",
35       });
36     }
37
38     // สร้าง Token สำหรับผู้ใช้
39     const token = jwt.sign({ userId: user._id }, process.env.jwt_secret, {
40       expiresIn: "1d",
41     });
42
43     res.send({
44       success: true,
45       message: "User has logged in successfully.",
46       data: token,
47     });
48   } catch (error) {
49     res.send({
50       success: false,
51       message: error.message,
52     });
53   }
54 });

```



```
1  /**
2   * เส้นทางสำหรับดึงรายละเอียดผู้ใช้คามาไว้อีด
3   * @name GET /api/users/get-current-user
4   * @function
5   * @memberof userRoutes
6   * @inner
7   * @param {Object} req - อีอบเจ็กต์ค่าขอ Express
8   * @param {string} req.body.userId - ไอดีของผู้ใช้
9   * @param {Object} res - อีอบเจ็กต์ค่าตอบ Express
10  * @returns {Object} สถานะและข้อมูลผู้ใช้หรือข้อความผิดพลาด
11 */
12 router.get("/get-current-user", authMiddleware, async (req, res) => {
13   try {
14     const user = await User.findById(req.body.userId).select("-password");
15     res.send({
16       success: true,
17       message: "User details fetched!",
18       data: user,
19     });
20   } catch (error) {
21     res.send({
22       success: false,
23       message: error.message,
24     });
25   }
26 });
27
28 module.exports = router;
```