Department of Computer Engineering ,
Faculty of Engineering ,
King Mongkut's Institute of Technology Ladkrabang

KMITL Cinema

Present to

Asst.Prof.Dr.Visit Hirankitti

Wiboon Promphanich

Submitted by

Phijak	Chanyawiwatkul	59010956
Pisith	Theplib	59010980
Pisit	Sriamonkitkul	59010981
Peerada	Cheepsomsong	59011068
Siripone	Ungrattanawaree	59011299

01076254 OBJECT-ORIENTED ANALYSIS AND DESIGN

Semester 1 Year 2018

Introduction

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา 01076254 OBJECT-ORIENTED ANALYSIS AND DESIGN ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ภาคเรียนที่ 1 ปีการศึกษา 2561 ซึ่งให้ทำการใช้ความรู้ที่ได้เรียนมาในวิชามาพัฒนาทำซอฟแวร์ที่ใช้บนเครื่องคอมพิวเตอร์ ระบบการจัดการภายในโรงภาพยนตร์

ส่วนประกอบภายในรายงานจะประกอบไปด้วย functional requirement, non-functional requirement, use case diagram, sequence diagram, class diagram, statechart diagram และ program source code

ทางคณะของผู้จัดทำหวังว่ารายงานฉบับนี้จะเป็นประโยชน์แก่ผู้ที่สนใจ หรือต้องการศึกษาไม่มากก็น้อย หากมีข้อผิดพลาดประการใด ขออภัยมา ณ ที่นี้

คณะผู้จัดทำ

Contents

Chapter I

Introduction and Requirement Specification

Functional Requirement

1. ผู้ใช้งานสามารถใช้งานระบบสมาชิกได้โดยทำการสมัครสมาชิก โดยใส่ข้อมูลของผู้ใช้ในการติดต่อ เช่น ชื่อผู้ใช้งาน, อีเมล, ชื่อ-นามสกุล, วันเดือนปีเกิด, รหัสผ่าน และ ยืนยันรหัสผ่าน
2. ผู้ใช้งานที่เคยสมัครสมาชิกไว้แล้ว สามารถเข้าสู่ระบบได้โดยใช้ ชื่อผู้ใช้งาน และรหัสผ่านที่ทำการสมัครกับระบบไว้ในการเข้ามาแก้ไขข้อมูลส่วนตัว ตามที่กล่าวไว้ในข้อที่ 1 ยกเว้นชื่อผู้ใช้งานที่ไม่สามารถแก้ไขได้
3. ผู้ใช้งานที่เข้าสู่ระบบอยู่ สามารถเลือกดูรายละเอียดโปรโมชั่นทั้งหมดที่มีในระบบ และเลือกใช้โปรโมชั่นดังกล่าวในการใช้เป็นส่วนลดหรือผลประโยชน์ในการซื้อ หรือจอง ตั๋วภาพยนตร์ได้
4. ผู้ใช้งานสามารถเรียกดูรายการของภาพยนตร์ทั้งหมดที่กำลังฉาย โดยมีรายละเอียดเป็นรอบฉาย, โรงที่ฉาย, ราคาต่อที่นั่ง และที่นั่งที่สามารถจองได้
5. ผู้ใช้งานที่เข้าสู่ระบบอยู่ สามารถเลือกภาพยนตร์และรอบฉายที่ต้องการซื้อหรือจองตั๋ว เพื่อทำการเลือกที่นั่งที่ว่างอยู่ โดยจะแสดงข้อมูลและราคาของภาพยนตร์ในรอบฉายนั้นๆ เมื่อเลือกที่นั่งที่ต้องการได้แล้วระบบจะคำนวณราคาที่นั่งทั้งหมดที่เลือก และสามารถทำการเลือกใช้โปรโมชั่นที่มีในระบบได้ถ้าผู้ใช้ต้องการ ในส่วนของโปรโมชั่นนั้นจะมีการจำกัดสิทธิ์ให้ 1 โปรโมชั่น/1ผู้ใช้งาน โดยจะตัดเงินจากระบบของผู้ใช้งาน ถ้าเงินในระบบมีไม่เพียงพอผู้ใช้งานสามารถเติมเงินเข้าสู่ระบบได้ เมื่อมีเงินในระบบเพียงพอระบบจะทำการชำระเงินจริง
6. ผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff) สามารถจำหน่ายตั๋วให้ลูกค้าที่มาซื้อที่เคาเตอร์ขายตั๋วได้ ซึ่งรายละเอียดการซื้อตั๋วจะเหมือนกับในข้อที่ 5 แต่ไม่สามารถเลือกใช้โปรโมชั่น และไม่สามารถชำระเงินผ่านระบบได้
7. ผู้ใช้งานที่เข้าสู่ระบบอยู่ สามารถยกเลิกตั๋วภาพยนตร์ที่ทำการจองไว้ ระบบจะทำการหักเงิน 10 เปอร์เซ็นต์ของค่าตั๋วภาพยนตร์ และทำการคืนเงินคงเหลือเข้าสู่ระบบของผู้ใช้งาน
8. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถเพิ่มโรงภาพยนต์ โดยมี หมายเลขโรงภาพยนต์, ขนาดของโรงภาพยนต์ และราคาพื้นฐานต่อที่นั่ง
9. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถแก้ไขรายละเอียดโรงภาพยนต์ตามที่กล่าวไว้ในข้อที่ 8
10. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถลบโรงภาพยนต์ที่เลือกได้

11. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถเพิ่มภาพยนตร์ โดยมีรายละเอียดเป็น ชื่อภาษาอังกฤษ, ชื่อภาษาไทย, ผู้กำกับ, นักแสดง, เนื้อเรื่องย่อ, ประเภทของภาพยนตร์, วันที่ฉาย, ระยะเวลาของภาพยนตร์, ภาพปกภาพยนตร์ และตัวอย่างวิดีโอภาพยนตร์
12. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถแก้ไขรายละเอียดภาพยนตร์ตามที่กล่าวไว้ในข้อที่ 11
13. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถลบภาพยนตร์ที่เลือกได้
14. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถเพิ่มรอบฉาย โดยมีรายละเอียด หมายเลขโรงภาพยนต์, ภาพยนตร์ที่ฉาย, ภาษา, บทบรรยาย, ราคาที่เพิ่มต่อที่นั่ง, เวลา และวันที่ฉาย
15. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถแก้ไขรายละเอียดรอบฉายตามที่กล่าวไว้ในข้อที่ 14
16. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถลบรอบฉายที่เลือกได้
17. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถเพิ่มโปรโมชั่น โดยมีรายละเอียดเป็น ชื่อโปรโมชั่น, คำอธิบาย, ส่วนลด และ วันหมดอายุ
18. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถแก้ไขรายละเอียดโปรโมชั่นตามที่กล่าวไว้ในข้อที่ 17
19. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถลบโปรโมชั่นที่เลือกได้
20. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถแก้ไขบทบาทของผู้ใช้งานในระบบเป็นพนักงาน (Staff) หรือลูกค้าที่เป็นสมาชิก (Member)
21. กรณีที่ต้องการลบผู้ใช้งานออกจากระบบ ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) สามารถลบผู้ใช้งานที่เลือกได้

ตารางการใช้งานของแต่ละบทบาท

Function	Non-Member	Member	Staff	Admin
สร้างบัญชีผู้ใช้งาน	✓			
แก้ไขบัญชีผู้ใช้งาน		✓		
เข้าสู่ระบบผู้ใช้งาน		✓	✓	✓
เพิ่มเงินเข้าสู่บัญชีของผู้ใช้		✓		
ดูและเลือกโปรโมชั่น		✓		
จองตั๋วออนไลน์		✓		
ยกเลิกการจองตั๋วออนไลน์		✓		
แสดงรอบของภาพยนต์ที่กำลังฉายอยู่	✓	✓	✓	✓
แสดงภาพยนต์ที่กำลังจะเข้าฉาย	✓	✓	✓	✓
จำหน่ายตั๋วผ่านหน้าเคาเตอร์			✓	
ตั้งค่าที่นั่งของโรงภาพยนต์				✓
เพิ่ม แก้ไข และลบโรงภาพยนต์				✓
เพิ่ม แก้ไข และลบรอบฉาย				✓
เพิ่ม แก้ไข และลบภาพยนตร์				✓
แก้ไข และลบผู้ใช้งาน				✓
เพิ่ม แก้ไข และลบโปรโมชั่น				✓

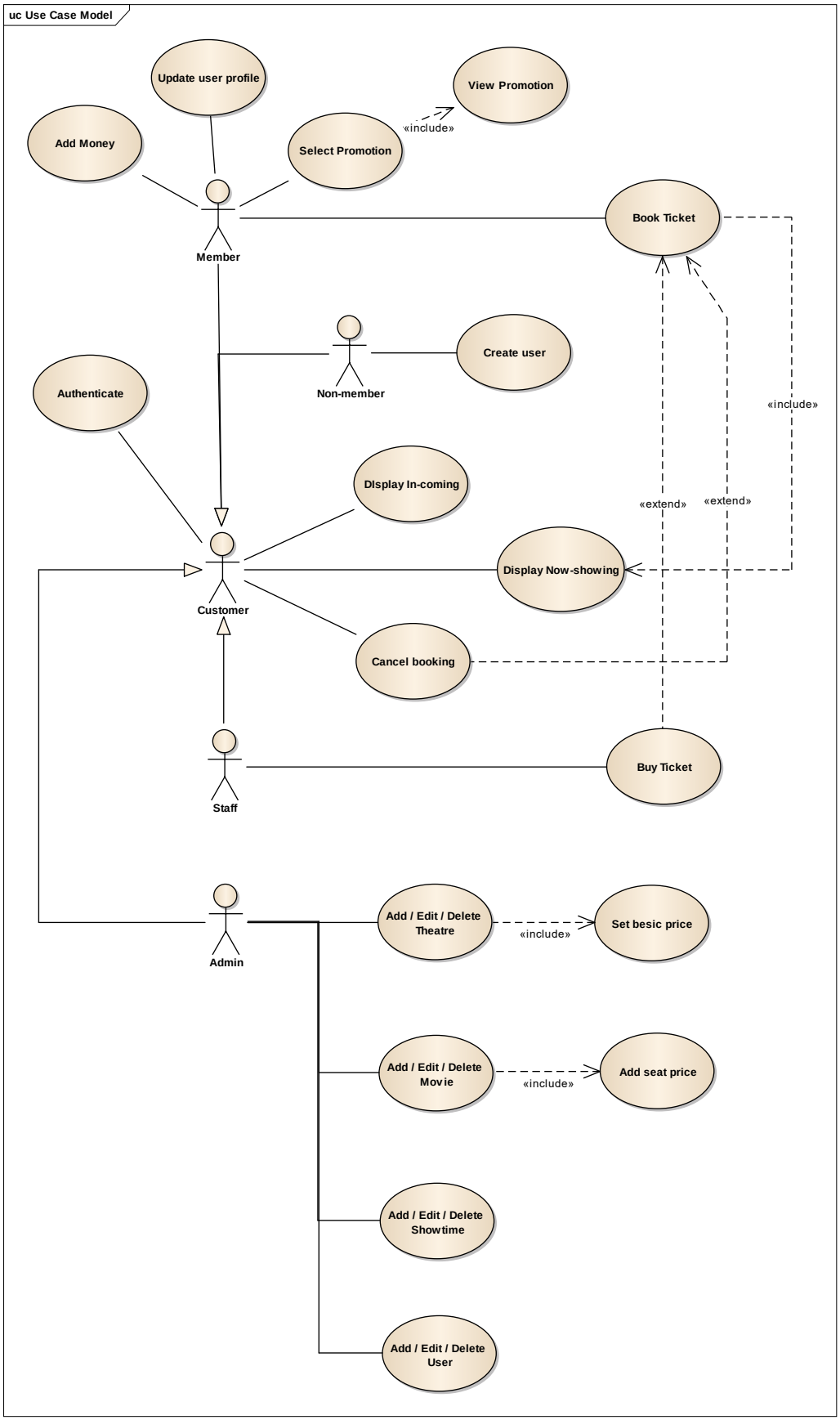
Non-functional Requirement

- การเก็บรักษาข้อมูล (Data retention)
    - ข้อมูลของผู้ใช้ในการติดต่อ เช่น ชื่อผู้ใช้งาน, รหัสผ่าน, ชื่อ-นามสกุล,วันเดือนปีเกิด และอีเมล ซึ่งข้อมูลที่ผู้ใช้กรอกเข้ามาจะถูกบันทึกไว้ในฐานข้อมูล
    - ผู้ใช้สามารถแก้ข้อมูลโดยต้องได้รับการอนุญาติให้เข้าถึงเมื่อรหัสผ่านที่ถูกต้อง
    - กรณีทำการเพิ่มโรงภาพยนต์ ระบบจะทำการตรวจสอบว่ามีหมายเลขโรงภาพยนต์ดังกล่าวถูกสร้างไว้ในระบบแล้วหรือไม่ ถ้าตรวจพบระบบจะแจ้งเตือนว่ามีการสร้างโรงภาพยนต์ซ้ำ
    - กรณีทำการแก้ไขรายละเอียดภาพยนตร์ ระบบจะทำการตรวจสอบว่าข้อมูลที่ถูกแก้นั้นจะไม่ไปซ้ำกับข้อมูลได้บันทึกไว้
    - กรณีทำการเพิ่มรอบฉาย ระบบจะทำการตรวจสอบว่าภายในโรงภาพยนต์นั้นสามารถเพิ่มเวลานี้เข้าไปได้หรือไม่ ถ้าตรวจพบว่ามีรอบฉายซ้ำกันระบบจะแจ้งเตือนว่ามีการสร้างรอบฉายซ้ำ
    - กรณีทำการแก้ไขรายละเอียดโปรโมชั่น ระบบจะทำการตรวจสอบว่าข้อมูลที่ถูกแก้นั้นจะไม่ไปซ้ำกับข้อมูลได้บันทึกไว้
    - กรณีทำการเลือก จะมีการส่งค่าไอดีที่เลือกไปใช้ในการทำงานต่อ
- ความเข้ากันได้ของแพลตฟอร์ม (Platform compatibility)
    - ผู้ใช้งานสามารถใช้แอพพลิเคชั่นที่เป็น Desktop Application บนระบบปฏิบัติการ Windows 10 และมีการเชื่อมต่อกับระบบเครือข่ายออนไลน์
- ความปลอดภัย (Security)
    - รหัสผ่านของผู้ใช้จะถูกเข้ารหัสไว้เพื่อความปลอดภัย
    - รูปแบบการกรอกข้อมูลจะต้องตรงกับรูปแบบที่กำหนดไว้ ถ้าไม่ตรงจะทำการบันทึกไม่ได้

Chapter 2

Analysis

Use Case diagram



Authenticate

- basic courses of events
    1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin), พนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) กรอกข้อมูลชื่อผู้ใช้งาน และรหัสผ่าน ลงในหน้าเข้าสู่ระบบ หลังจากนั้นกด “เข้าสู่ระบบ”
    2. ระบบจะทำการตรวจสอบชื่อผู้ใช้งานนี้ในฐานข้อมูล
    3. ระบบจะทำการตรวจสอบรหัสผ่านนี้ในฐานข้อมูล
    4. ระบบจะทำการตรวจสอบว่าเป็นผู้ใช้งานประเภทไหน
    5. ระบบจะทำการเข้าสู่ระบบ
- alternative courses of events

  <u>กรณีไม่พบชื่อผู้ใช้งานในฐานข้อมูล</u>
    1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin), พนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) กรอกข้อมูลชื่อผู้ใช้งาน และรหัสผ่าน ลงในหน้าเข้าสู่ระบบ หลังจากนั้นกด “เข้าสู่ระบบ”
    2. ระบบจะทำการตรวจสอบชื่อผู้ใช้งานนี้ในฐานข้อมูลว่าพบหรือไม่
    3. ถ้าระบบพบ จะทำงานขั้นต่อไป
    4. ถ้าระบบไม่พบ จะแจ้งเตือนว่าไม่พบชื่อผู้ใช้งานนี้ จะเข้าระบบสู่ระบบไม่ได้

  <u>กรณีไม่พบชื่อผู้ใช้งานในฐานข้อมูล</u>
    1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin), พนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) กรอกข้อมูลชื่อผู้ใช้งาน และรหัสผ่าน ลงในหน้าเข้าสู่ระบบ หลังจากนั้นกด “เข้าสู่ระบบ”
    2. ระบบจะทำการตรวจสอบชื่อผู้ใช้งานนี้ในฐานข้อมูล
    3. ระบบจะทำการตรวจสอบรหัสผ่านนี้ในฐานข้อมูลว่าพบหรือไม่
    4. ถ้าระบบพบ จะทำงานขั้นต่อไป
    5. ถ้าระบบไม่พบ จะแจ้งเตือนว่ารหัสผ่านไม่ถูกต้อง จะเข้าระบบสู่ระบบไม่ได้

Create user

- basic courses of events
    1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่ไม่ได้เป็นสมาชิก(Non-member) กรอกข้อมูล ชื่อผู้ใช้งาน, อีเมล, ชื่อ-นามสกุล, วันเดือนปีเกิด, รหัสผ่าน และ ยืนยันรหัสผ่าน ลงในหน้าสมัครสมาชิก หลังจากนั้นกด “สมัครสมาชิก”
    2. ระบบจะทำการตรวจสอบ รหัสผ่าน และ ยืนยันรหัสผ่าน
    3. ระบบจะทำการตรวจสอบ ชื่อผู้ใช้งาน ในฐานข้อมูล
    4. ระบบจะทำการตรวจสอบ อีเมล ในฐานข้อมูล
    5. ระบบจะทำการแสดงว่าทำการสมัครสมาชิกเรียบร้อย
- alternative courses of events

<u>กรณีรหัสผ่านไม่ตรงกับยืนยันรหัสผ่าน</u>

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่ไม่ได้เป็นสมาชิก(Non-member) กรอกข้อมูล ชื่อผู้ใช้งาน, อีเมล, ชื่อ-นามสกุล, วันเดือนปีเกิด, รหัสผ่าน และ ยืนยันรหัสผ่าน ลงในหน้าสมัครสมาชิก หลังจากนั้นกด “สมัครสมาชิก”
2. ระบบจะทำการตรวจสอบว่า รหัสผ่าน และ ยืนยันรหัสผ่าน ตรงกันหรือไม่
3. ถ้าตรง ระบบจะทำงานขั้นต่อไป
4. ถ้าไม่ตรง จะแจ้งเตือนว่ารหัสผ่านไม่ตรงกัน จะทำการสมัครสมาชิกไม่ได้

<u>กรณีไม่พบชื่อผู้ใช้งานในฐานข้อมูล</u>

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่ไม่ได้เป็นสมาชิก(Non-member) กรอกข้อมูล ชื่อผู้ใช้งาน, อีเมล, ชื่อ-นามสกุล, วันเดือนปีเกิด, รหัสผ่าน และ ยืนยันรหัสผ่าน ลงในหน้าสมัครสมาชิก หลังจากนั้นกด “สมัครสมาชิก”
2. ระบบจะทำการตรวจสอบ รหัสผ่าน และ ยืนยันรหัสผ่าน
3. ระบบจะทำการตรวจสอบชื่อผู้ใช้งานนี้ในฐานข้อมูลว่าพบหรือไม่
4. ถ้าระบบพบ จะทำงานขั้นต่อไป
5. ถ้าระบบไม่พบ จะแจ้งเตือนว่าไม่พบชื่อผู้ใช้งานนี้ จะทำการสมัครสมาชิกไม่ได้

<u>กรณีไม่พบอีเมลในฐานข้อมูล</u>

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่ไม่ได้เป็นสมาชิก(Non-member) กรอกข้อมูล ชื่อผู้ใช้งาน, อีเมล, ชื่อ-นามสกุล, วันเดือนปีเกิด, รหัสผ่าน และ ยืนยันรหัสผ่าน ลงในหน้าสมัครสมาชิก หลังจากนั้นกด “สมัครสมาชิก”
2. ระบบจะทำการตรวจสอบ รหัสผ่าน และ ยืนยันรหัสผ่าน
3. ระบบจะทำการตรวจสอบ ชื่อผู้ใช้งาน ในฐานข้อมูล
4. ระบบจะทำการตรวจสอบว่า อีเมล นี้ในฐานข้อมูลพบหรือไม่
5. ถ้าระบบพบ จะแจ้งเตือนว่าอีเมลนี้มีคนใช้แล้ว จะทำการสมัครสมาชิกไม่ได้
6. ถ้าระบบไม่พบ จะทำงานขั้นต่อไป

Book Ticket

- basic courses of events

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกภาพยนตร์ ในหน้าแสดงภาพยนตร์ที่ฉายอยู่ตอนนี้
2. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของภาพยนตร์ที่เลือก
3. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) กด “ถัดไป”
4. ระบบจะทำการตรวจสอบ และแสดงรอบฉายภาพยนตร์ของวันนั้น
5. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกรอบฉายภาพยนตร์
6. ระบบจะทำการตรวจสอบ และแสดงที่นั่งภายในโรงภาพยนตร์
7. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกที่นั่งที่ต้องการ
8. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของภาพยนตร์ที่เลือก, รายละเอียดของที่นั่ง และค่าตั๋วภาพยนตร์ที่ต้องจ่าย
9. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) จะสามารถทำการเลือกโปรโมชั่นได้
10. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของโปรโมชั่นที่เลือก
11. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) กด “ชำระเงิน”
12. ระบบจะทำการตรวจสอบ แสดงเงินในระบบของผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member)
13. ระบบจะทำการตรวจสอบเงินในระบบ
14. ระบบจะทำการจอง

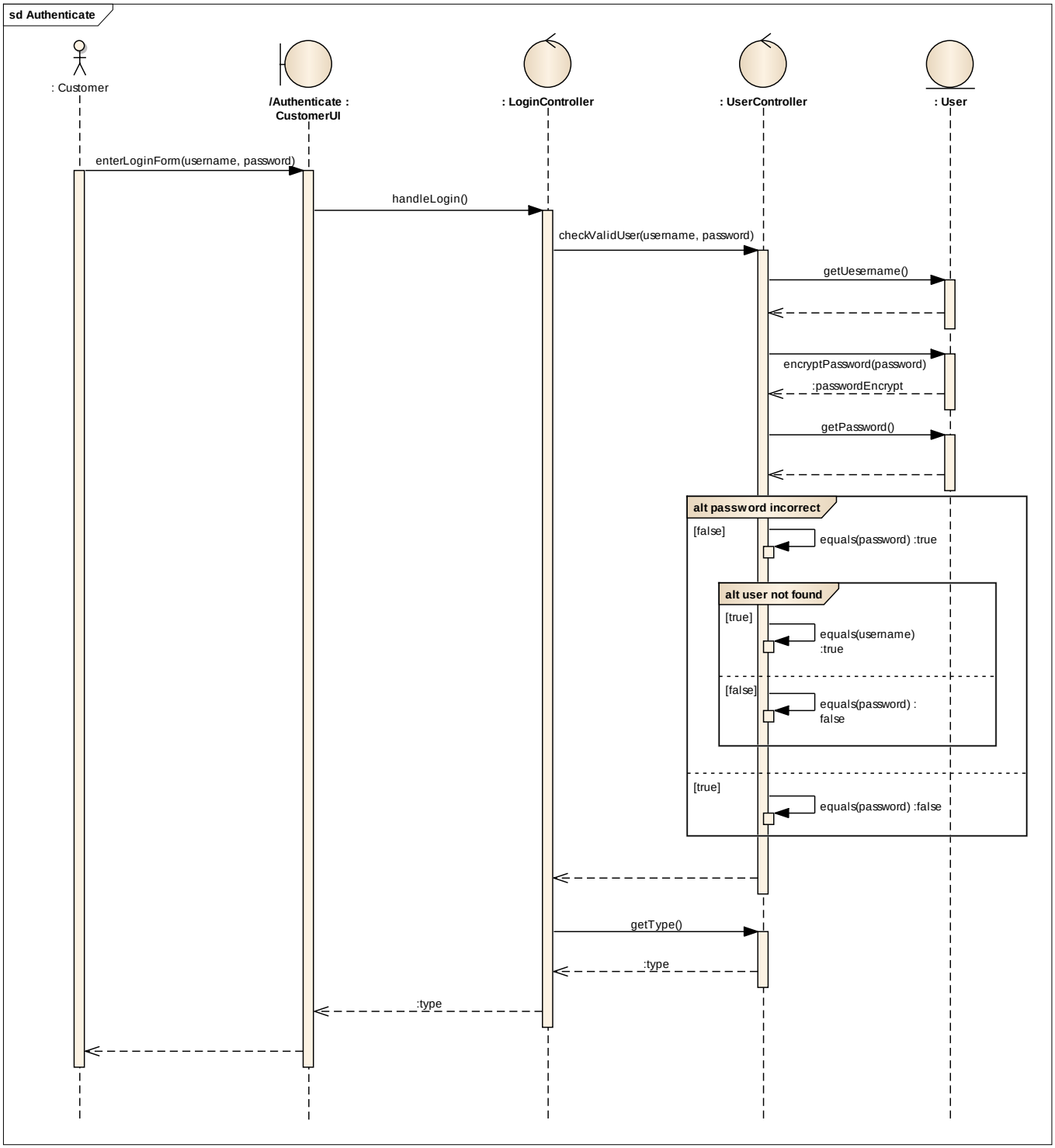
- alternative courses of events

<u>กรณีเงินในระบบไม่เพียงพอ</u>

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกภาพยนตร์ ในหน้าแสดงภาพยนตร์ที่ฉายอยู่ตอนนี้
2. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของภาพยนตร์ที่เลือก
3. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) กด “ถัดไป”
4. ระบบจะทำการตรวจสอบ และแสดงรอบฉายภาพยนตร์ของวันนั้น
5. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกรอบฉายภาพยนตร์
6. ระบบจะทำการตรวจสอบ และแสดงที่นั่งภายในโรงภาพยนตร์
7. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกที่นั่งที่ต้องการ
8. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของภาพยนตร์ที่เลือก, รายละเอียดของที่นั่ง และค่าตั๋วภาพยนตร์ที่ต้องจ่าย
9. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) จะสามารถทำการเลือกโปรโมชั่นได้
10. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของโปรโมชั่นที่เลือก
11. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) กด “ชำระเงิน”
12. ระบบจะทำการตรวจสอบ แสดงเงินในระบบของผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member)
13. ระบบจะทำการตรวจสอบเงินในระบบว่ามีเพียงพอหรือไม่
15. ถ้าเพียงพอ จะทำงานขั้นต่อไป
16. ถ้าไม่เพียงพอ จะแจ้งเตือนว่าเงินในระบบมีไม่เพียงพอ จะทำการจองตั๋วภาพยนตร์ไม่ได้

Sequence diagram

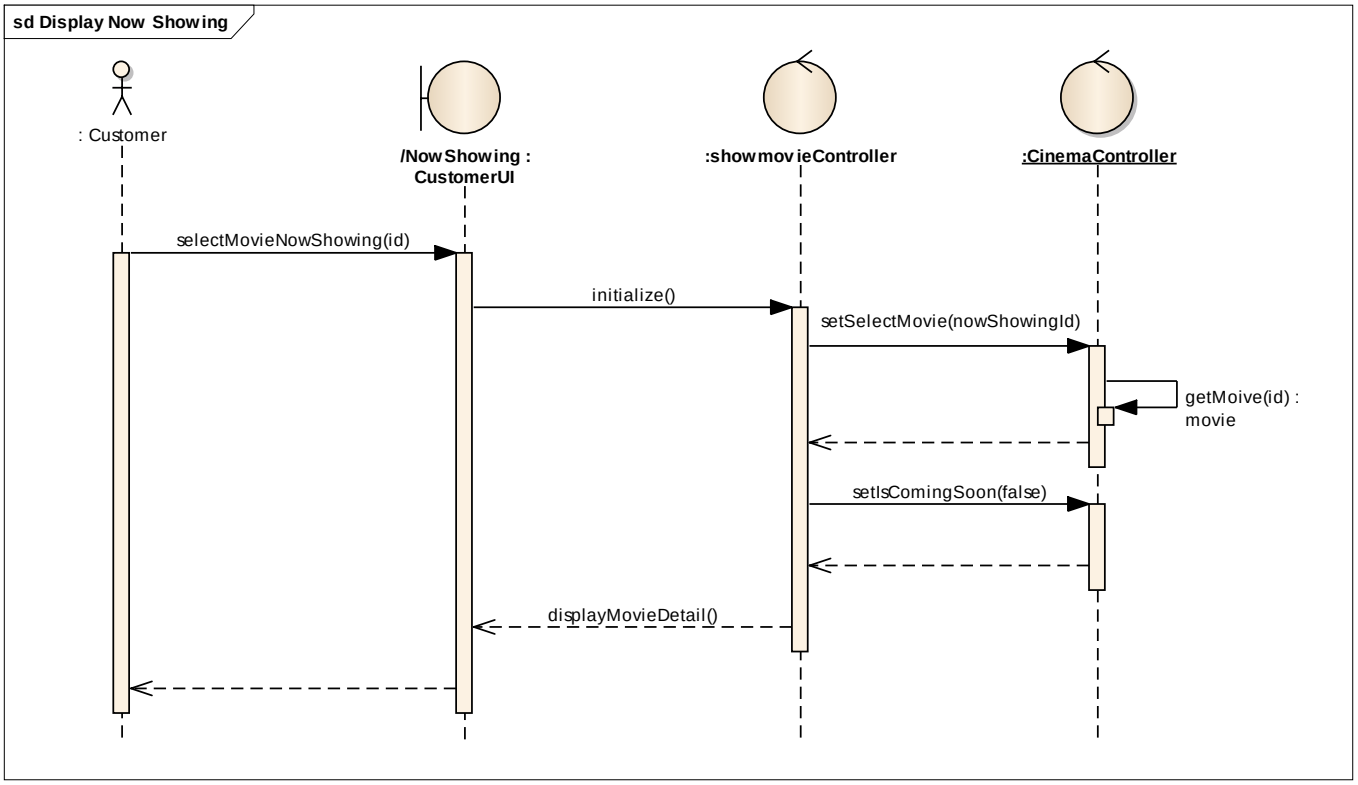
Login ใช้เข้าสู่ระบบ โดยผู้ใช้งานประเภทต่างๆ



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin), พนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) กรอกข้อมูลชื่อผู้ใช้งาน และรหัสผ่าน ลงในหน้าเข้าสู่ระบบ หลังจากนั้นกด “เข้าสู่ระบบ”
2. ระบบจะทำการตรวจสอบว่าชื่อผู้ใช้งานนี้มีอยู่ในฐานข้อมูลแล้วหรือไม่
   2.1. ถ้าพบ จะทำงานขั้นต่อไป
   2.2. ถ้าไม่พบ จะแจ้งเตือนว่าไม่พบชื่อผู้ใช้งานนี้ ให้ทำการกรอกใหม่
3. ระบบจะทำการตรวจสอบว่ารหัสผ่านนี้มีอยู่ในฐานข้อมูลแล้วหรือไม่
   3.1. ถ้าพบ จะทำงานขั้นต่อไป
   3.2. ถ้าไม่พบ จะแจ้งเตือนว่ารหัสผ่านไม่ถูกต้อง ให้ทำการกรอกใหม่
4. ถ้าระบบพบว่าในฐานข้อมูลมีก็จะทำการตรวจสอบว่าเป็นผู้ใช้งานประเภทไหน
5. ระบบจะทำการเข้าสู่ระบบ

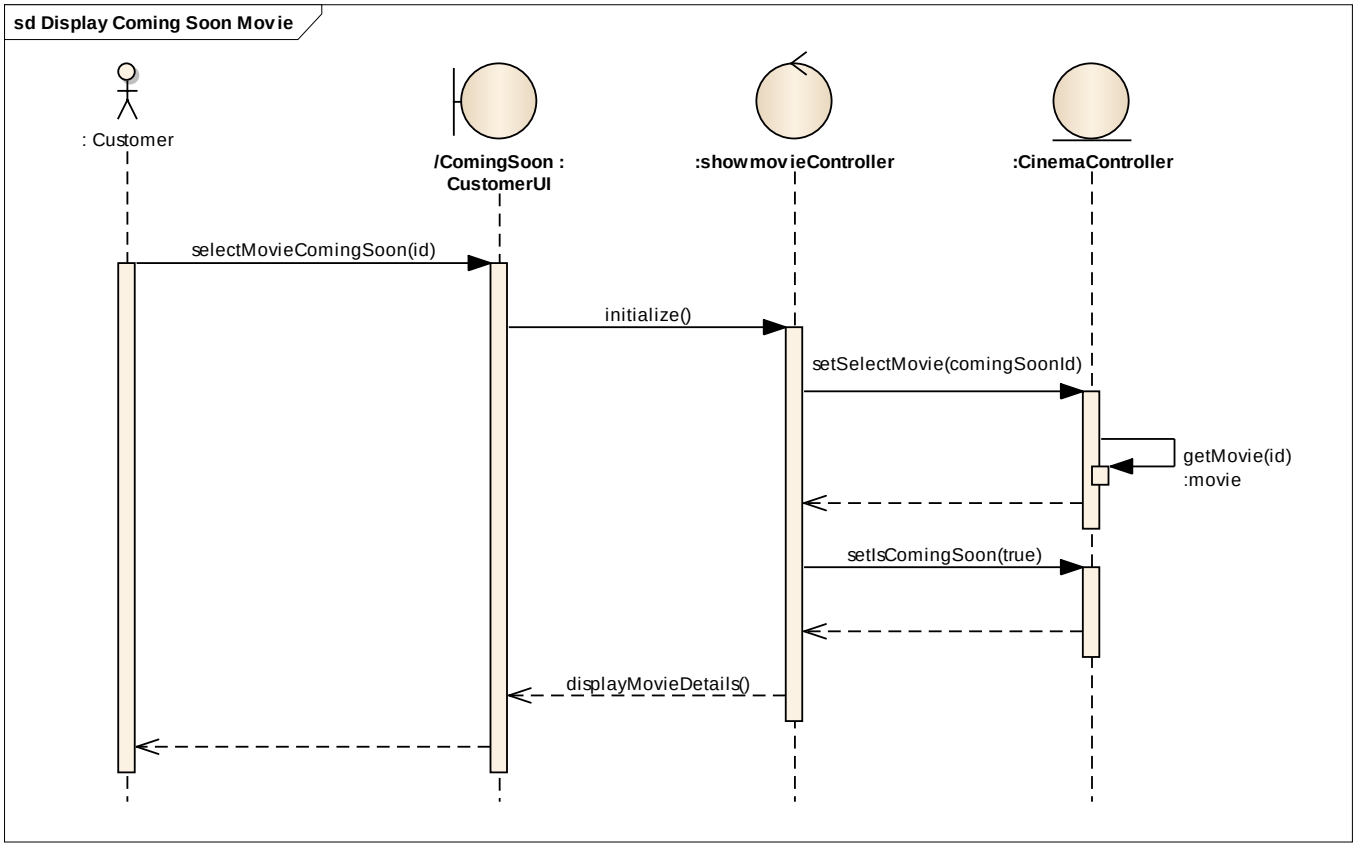
แสดงภาพยนตร์ที่ฉายอยู่ตอนนี้



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานเลือกภาพยนตร์
2. ระบบจะทำการตรวจสอบภาพยนตร์ที่เลือก และจะแสดงรายละเอียดของภาพยนตร์

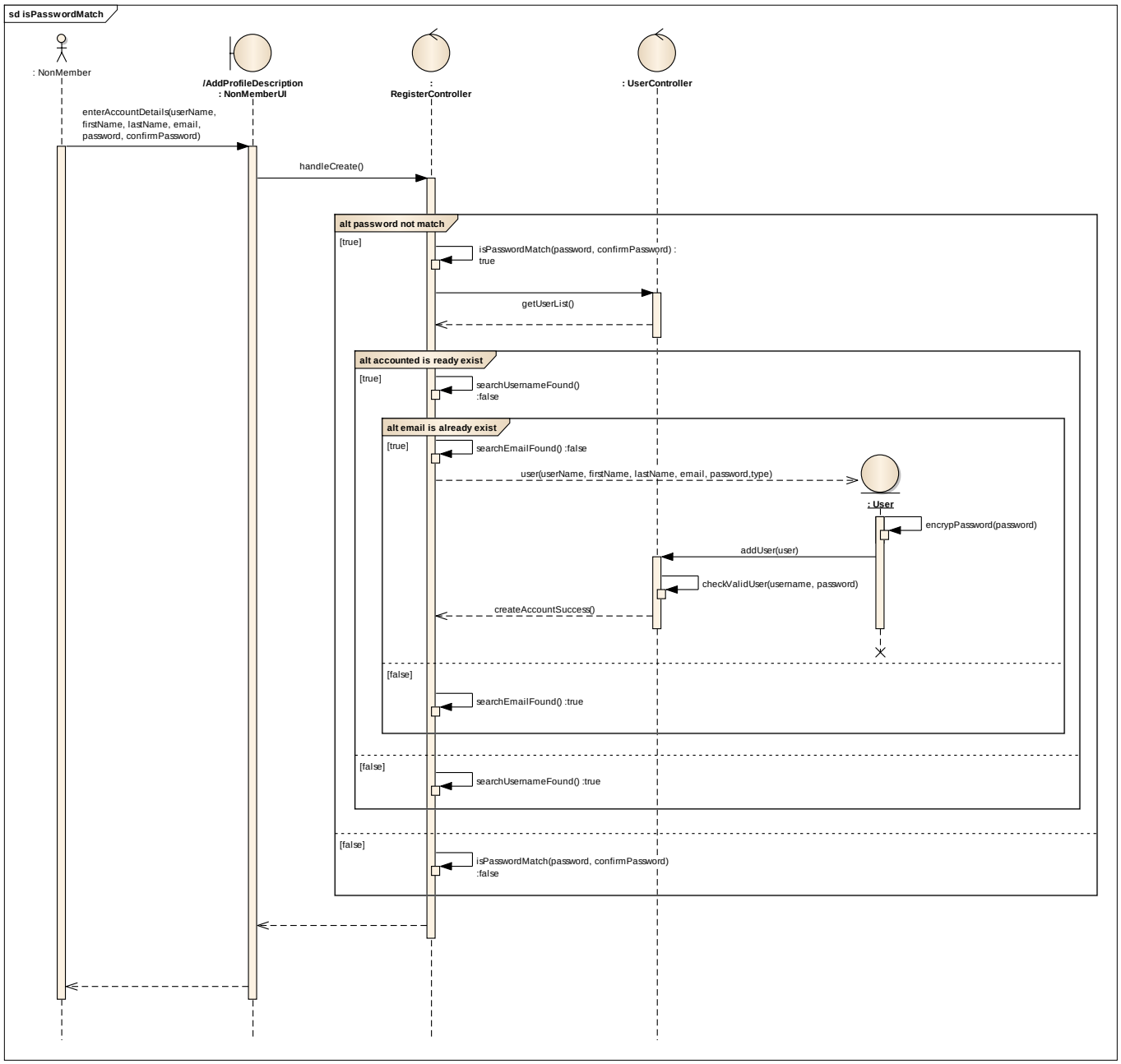
แสดงภาพยนตร์ที่กำลังจะเข้าฉาย



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานเลือกภาพยนตร์
2. ระบบจะทำการตรวจสอบภาพยนตร์ที่เลือก และจะแสดงรายละเอียดของภาพยนตร์

การสมัครเป็นสมาชิก

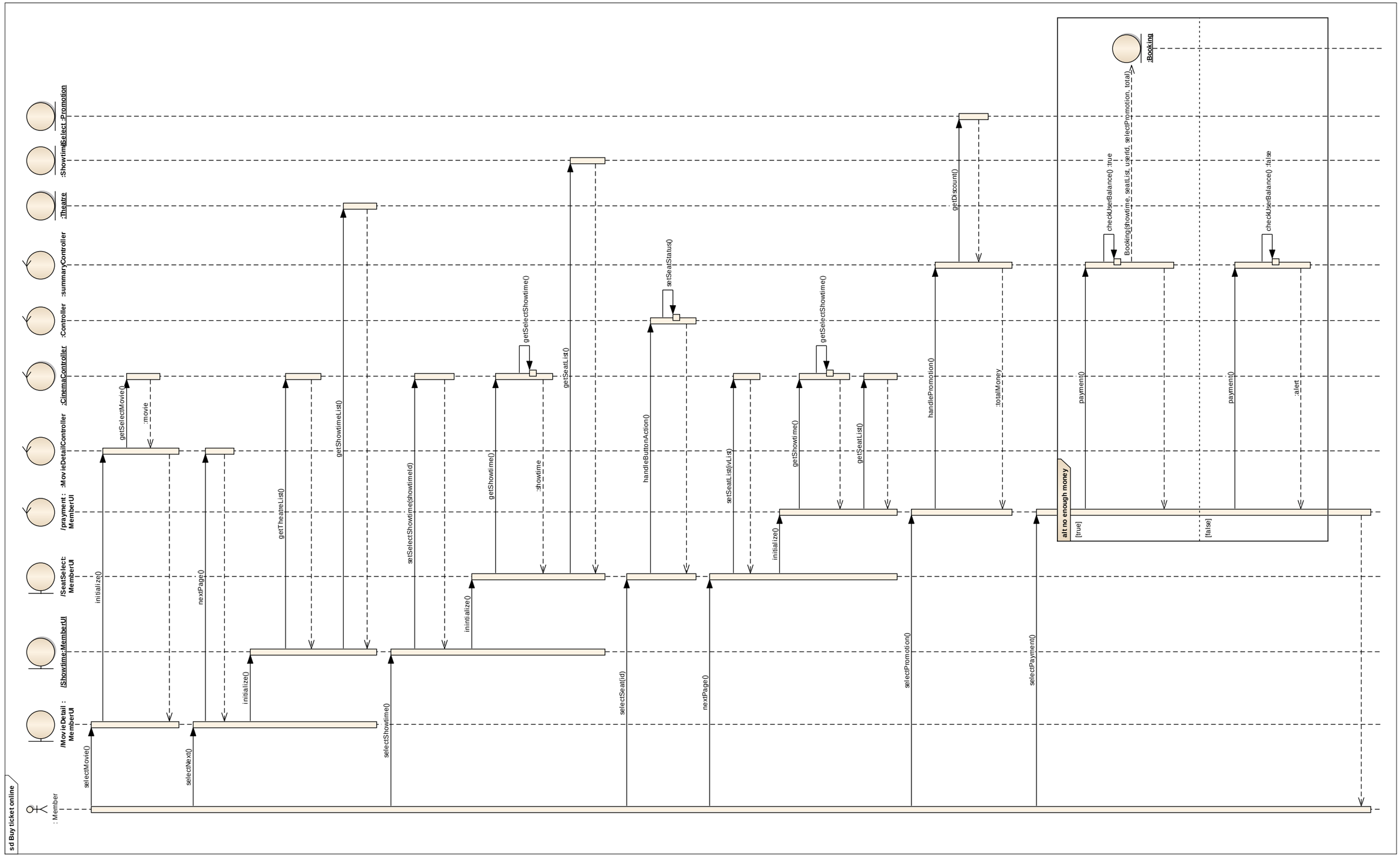


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่ไม่ได้เป็นสมาชิก(Non-member) กรอกข้อมูล ชื่อผู้ใช้งาน, อีเมล, ชื่อ-นามสกุล, วันเดือนปีเกิด, รหัสผ่าน และ ยืนยันรหัสผ่าน ลงในหน้าสมัครสมาชิก หลังจากนั้นกด “สมัครสมาชิก”
2. ระบบจะทำการตรวจสอบว่า รหัสผ่าน และ ยืนยันรหัสผ่าน ตรงกันหรือไม่
   2.1. ถ้าตรง จะทำงานขั้นต่อไป
   2.2. ถ้าไม่ตรง จะแจ้งเตือนว่าใส่รหัสผ่านไม่ตรงกัน ให้ทำการกรอกใหม่

3. ระบบจะทำการตรวจสอบว่า ชื่อผู้ใช้งาน นี้มีอยู่ในฐานข้อมูลแล้วหรือไม่
   3.1. ถ้าพบ จะแจ้งเตือนว่าชื่อนี้มีคนใช้แล้ว ให้ทำการกรอกใหม่
   3.2. ถ้าไม่พบ จะทำงานขั้นต่อไป
4. ระบบจะทำการตรวจสอบว่า อีเมล นี้มีอยู่ในฐานข้อมูลแล้วหรือไม่
   4.1. ถ้าพบ จะแจ้งเตือนว่าอีเมลนี้มีคนใช้แล้ว ให้ทำการกรอกใหม่
   4.2. ถ้าไม่พบ จะทำงานขั้นต่อไป
5. ระบบจะทำการแสดงว่าทำการสมัครสมาชิกเรียบร้อย

การจองรอบฉาย



มีขั้นตอนการทำงานดังนี้

14. ผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) เลือกภาพยนตร์ ในหน้าแสดงภาพยนตร์ที่ฉายอยู่ตอนนี้ หลังจากนั้นกด “BUY NOW”
15. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของภาพยนตร์ที่เลือก
16. ผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) กด “ถัดไป”
17. ระบบจะทำการตรวจสอบ และแสดงรอบฉายภาพยนตร์ของวันนั้น
18. ผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) เลือกรอบฉายภาพยนตร์
19. ระบบจะทำการตรวจสอบ และแสดงที่นั่งภายในโรงภาพยนตร์
20. ผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff) และ ลูกค้าที่เป็นสมาชิก (Member) เลือกที่นั่งที่ต้องการ
21. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของภาพยนตร์ที่เลือก, รายละเอียดของที่นั่ง และค่าตั๋วภาพยนตร์ที่ต้องจ่าย

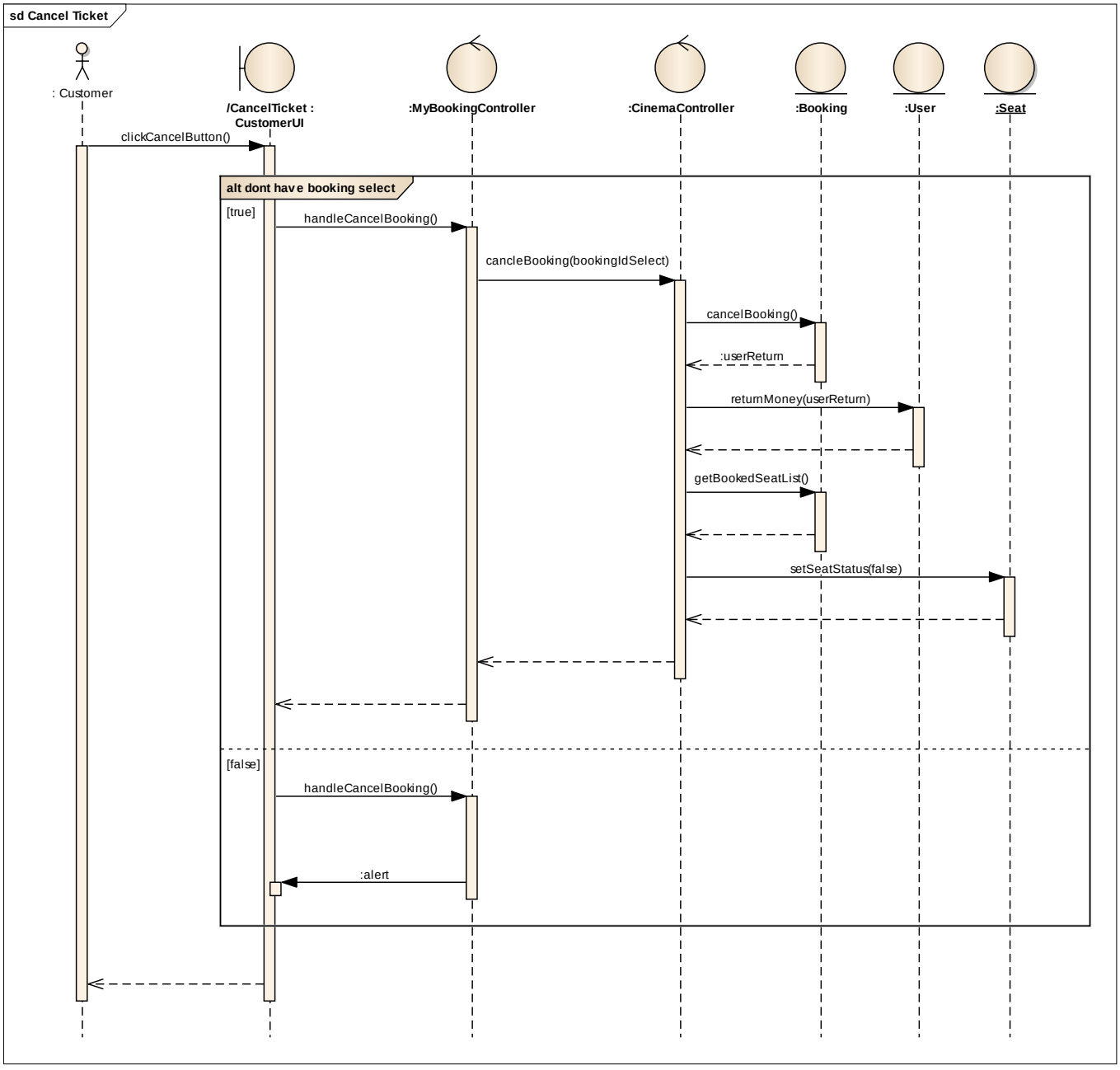
<u>กรณีผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff)</u>

22. ถ้าผู้ใช้งานที่มีบทบาทเป็นพนักงาน (Staff) กด “ชำระเงิน”
23. ระบบจะทำการจอง

<u>กรณีผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member)</u>

17. ถ้าผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) จะสามารถทำการเลือกโปรโมชั่นได้
18. ระบบจะทำการตรวจสอบ และแสดงรายละเอียดของโปรโมชั่นที่เลือก
19. ถ้าผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) กด “ชำระเงิน”
20. ระบบจะทำการตรวจสอบ แสดงเงินในระบบของผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member)
21. ระบบจะทำการตรวจสอบว่าเงินในระบบเพียงพอในการชำระเงินหรือไม่
    - 21.1. ถ้าเพียงพอ จะทำขั้นตอนต่อไป
    - 21.2. ถ้าไม่เพียงพอ จะแจ้งเตือนว่า กรุณาเติมเงิน
22. ระบบจะทำการจอง

การยกเลิกการจอง

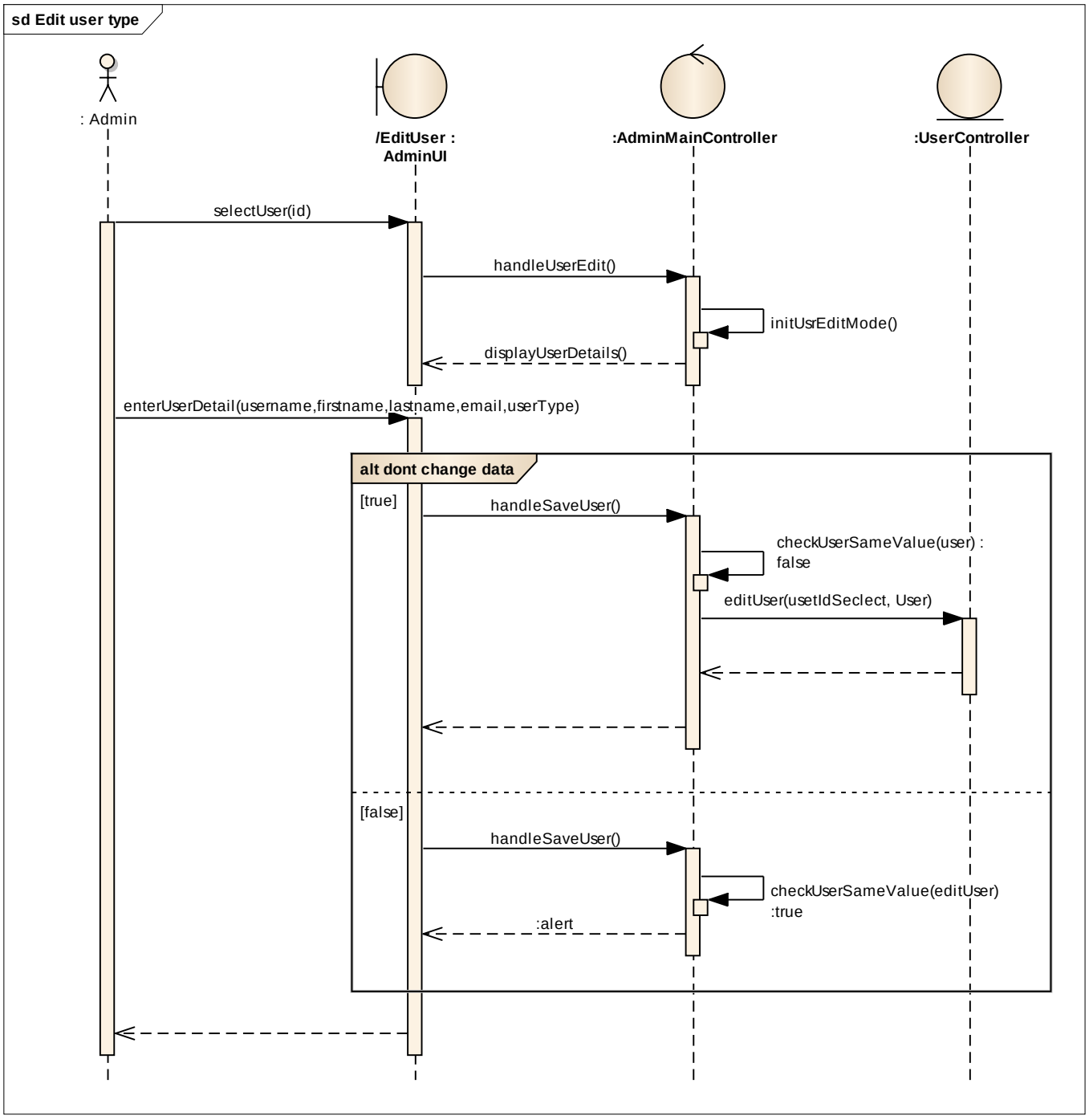


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นลูกค้าที่เป็นสมาชิก (Member) เลือกรอบฉายภาพยนตร์ ในหน้าแสดงรอบฉายภาพยนตร์ หลังจากนั้นคลิกขวาแล้วเลือก “cancel”

2. ระบบจะทำการตรวจสอบรอบฉายภาพยนตร์ที่เลือกว่าพบหรือไม่
   2.1. ถ้าพบ จะทำงานขั้นต่อไป
   2.2. ถ้าไม่พบ จะแจ้งเตือนว่า ไม่พบรอบฉายที่เลือก ให้ทำการเลือกใหม่
3. ระบบจะทำการหักเงิน 10 เปอร์เซ็นต์ของค่าตั๋วภาพยนตร์ และทำการคืนเงินคงเหลือเข้าสู่ระบบของผู้ใช้งาน
4. ระบบจะทำการยกเลิกรอบฉายภาพยนตร์ และแสดงว่าทำการยกเลิกรอบฉายภาพยนตร์เรียบร้อย

การแก้ไขประเภทของผู้ใช้งาน

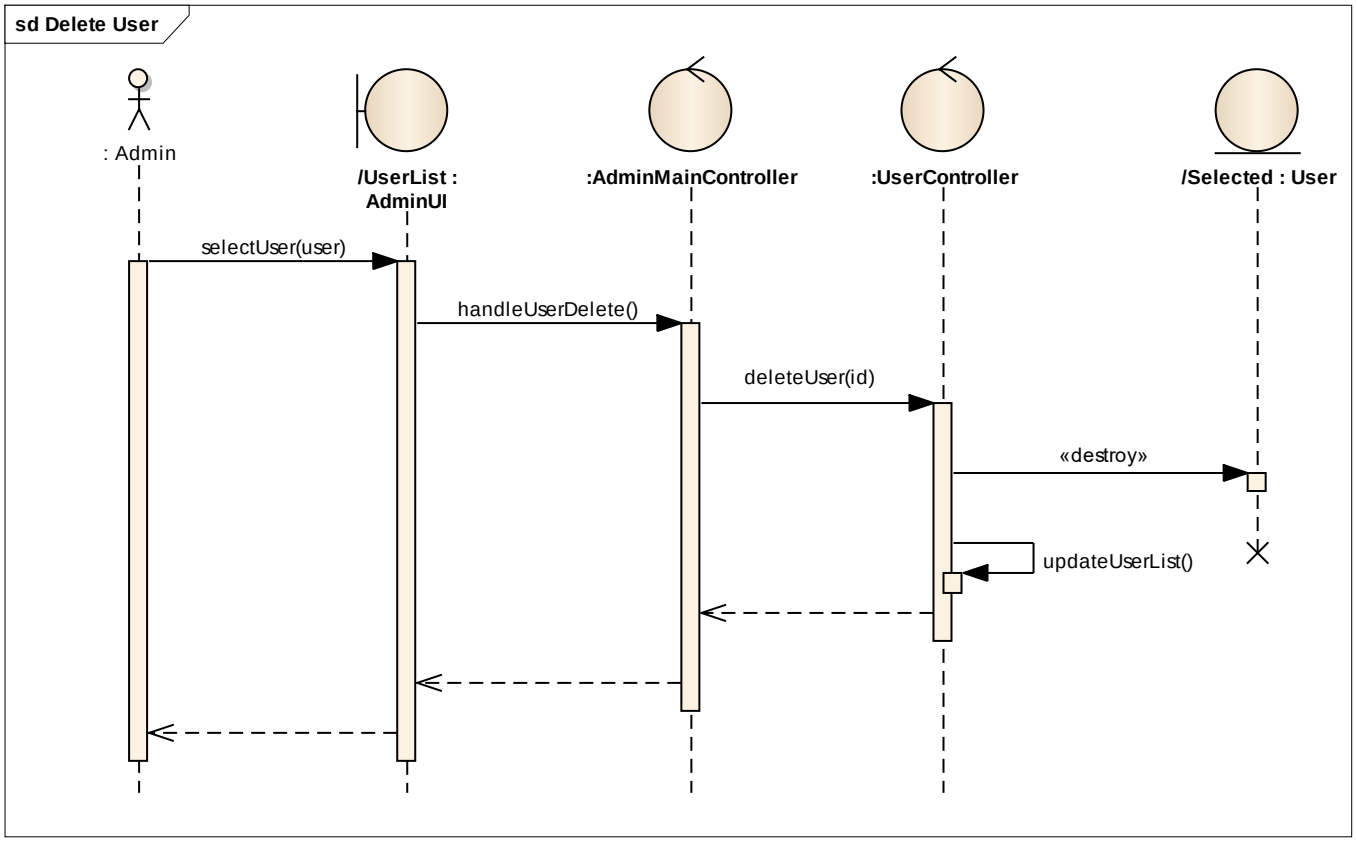


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกผู้ใช้งาน
2. ระบบจะทำการตรวจสอบผู้ใช้งานที่เลือก และแสดงรายละเอียดของผู้ใช้งาน

3. แก้ไขรายละเอียดซึ่งที่สามารถแก้ไขได้ ได้แก่ ชื่อผู้ใช้งาน, ชื่อ-นามสกุล, อีเมล และประเภทของผู้ใช้งาน ลงในหน้าเพิ่ม แก้ไข และลบผู้ใช้งาน หลังจากนั้นกด “Save edit”
4. ระบบจะทำการตรวจสอบว่ามีการแก้ไขข้อมูลหรือไม่
   4.1. ถ้าแก้ไข จะทำงานขั้นต่อไป
   4.2. ถ้าไม่แก้ไข จะไม่ทำการแก้ไขข้อมูล
5. ระบบจะทำการแก้ไขข้อมูลผู้ใช้งาน และแสดงว่าทำการแก้ไขผู้ใช้งานเรียบร้อย

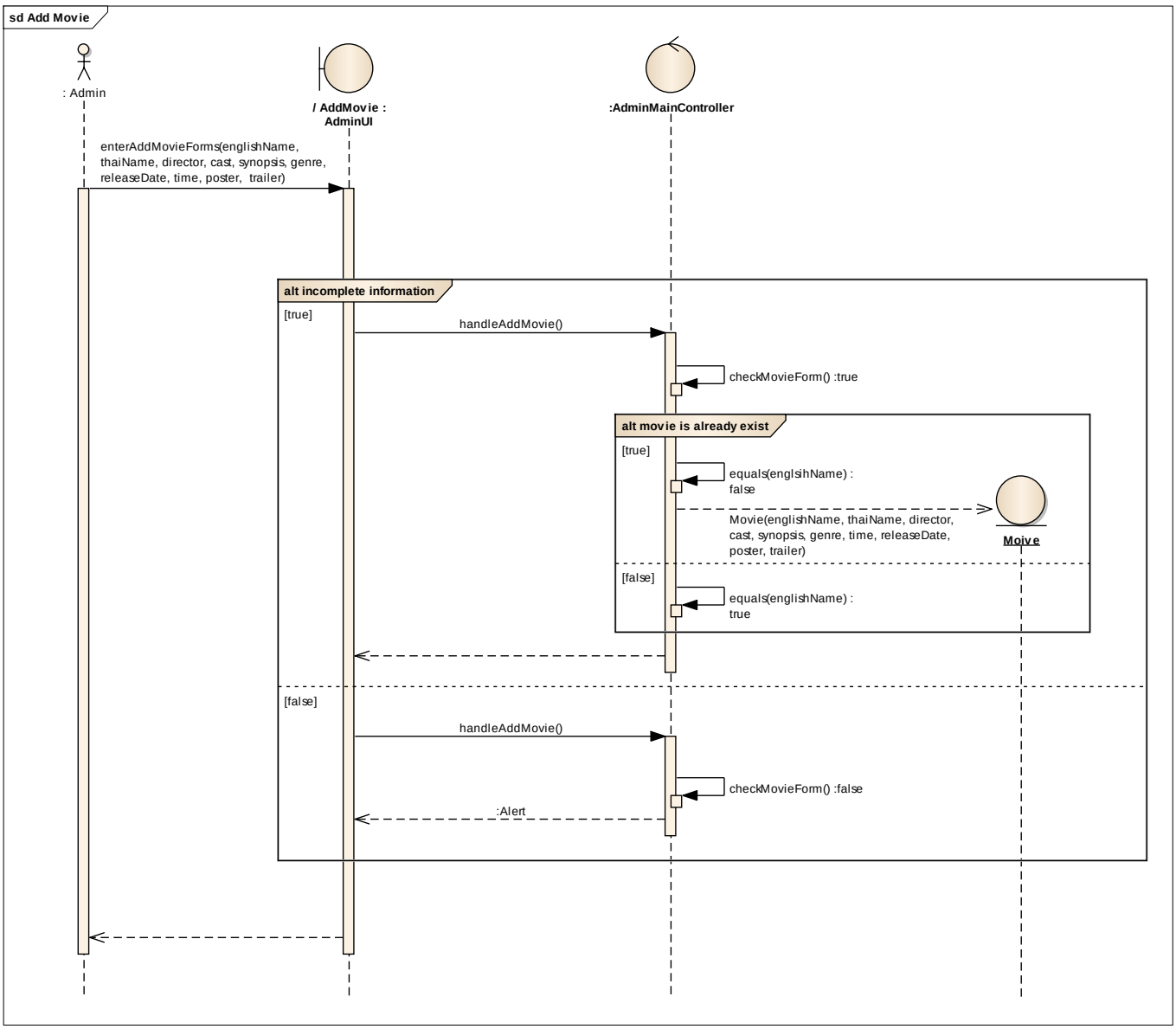
การลบผู้ใช้งาน



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกผู้ใช้งาน ในหน้าเพิ่ม แก้ไข และลบผู้ใช้งาน หลังจากนั้นคลิกขวาแล้วเลือก “delete”
2. ระบบจะทำการตรวจสอบผู้ใช้งานที่เลือก และทำการลบ
3. ระบบจะทำการลบผู้ใช้งาน และแสดงว่าทำการลบผู้ใช้งานเรียบร้อย

การเพิ่มภาพยนตร์

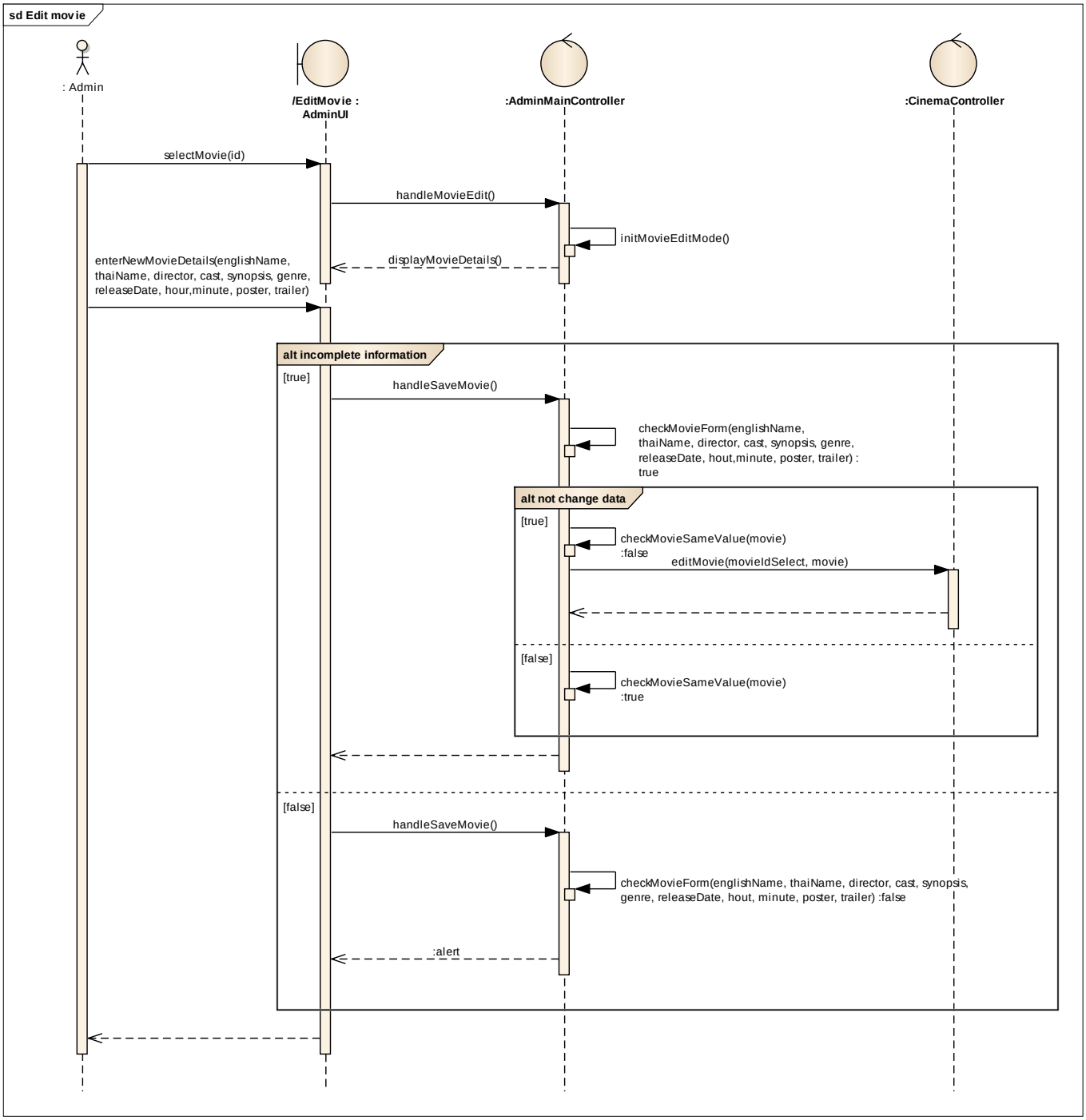


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) กรอกข้อมูล ชื่อภาษาอังกฤษ, ชื่อภาษาไทย, ผู้กำกับ, นักแสดง, เนื้อเรื่องย่อ, ประเภทของภาพยนตร์, วันที่ฉาย, ระยะเวลาของภาพยนตร์, ภาพปกภาพยนตร์ และตัวอย่างวิดีโอภาพยนตร์ ลงในหน้าเพิ่ม แก้ไข และลบภาพยนตร์ หลังจากนั้นกด “Add”
2. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   2.1. ถ้าถูก จะทำงานขั้นต่อไป
   2.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้กรอกข้อมูลใหม

3. ระบบจะทำการตรวจสอบว่ามีชื่อภาพยนตร์ภาษาอังกฤษเรื่องนี้อยู่ในฐานข้อมูลแล้วหรือไม่
   3.1. ถ้าพบ จะแจ้งเตือนว่า ภาพยนตร์นี้มีแล้ว ให้ทำการกรอกใหม่
   3.2. ถ้าไม่พบ จะทำงานขั้นต่อไป
4. ระบบจะทำการเพิ่มภาพยนตร์ และแสดงว่าทำการเพิ่มภาพยนตร์เรียบร้อย

การแก้ไขภาพยนตร์

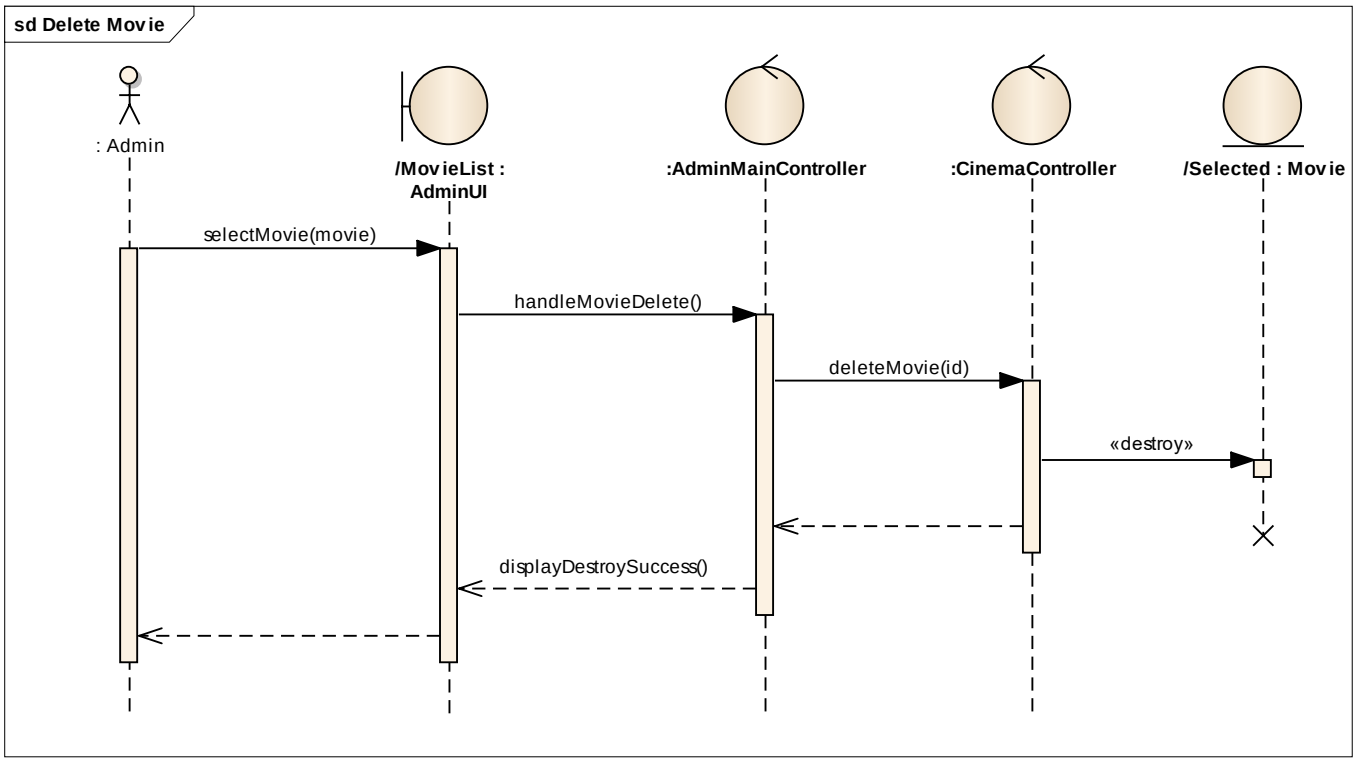


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกภาพยนตร์
2. ระบบจะทำการตรวจสอบภาพยนตร์ที่เลือก และแสดงรายละเอียดของภาพยนตร์

3. แก้ไขรายละเอียดซึ่งที่สามารถแก้ไขได้ ได้แก่ ชื่อภาษาอังกฤษ, ชื่อภาษาไทย, ผู้กำกับ, นักแสดง, เนื้อเรื่องย่อ, ประเภทของภาพยนตร์, วันที่ฉาย, ระยะเวลาของภาพยนตร์, ภาพปกภาพยนตร์ และตัวอย่างวิดีโอภาพยนตร์ ลงในหน้าเพิ่ม แก้ไข และลบภาพยนตร์ หลังจากนั้นกด “Save edit”
4. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   4.1. ถ้าถูก จะทำงานขั้นต่อไป
   4.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
5. ระบบจะทำการตรวจสอบว่ามีการแก้ไขข้อมูลหรือไม่
   5.1. ถ้าแก้ไข จะทำงานขั้นต่อไป
   5.2. ถ้าไม่แก้ไข จะไม่ทำการแก้ไขข้อมูล
6. ระบบจะทำการแก้ไขข้อมูลภาพยนตร์ และแสดงว่าทำการแก้ไขภาพยนตร์เรียบร้อย

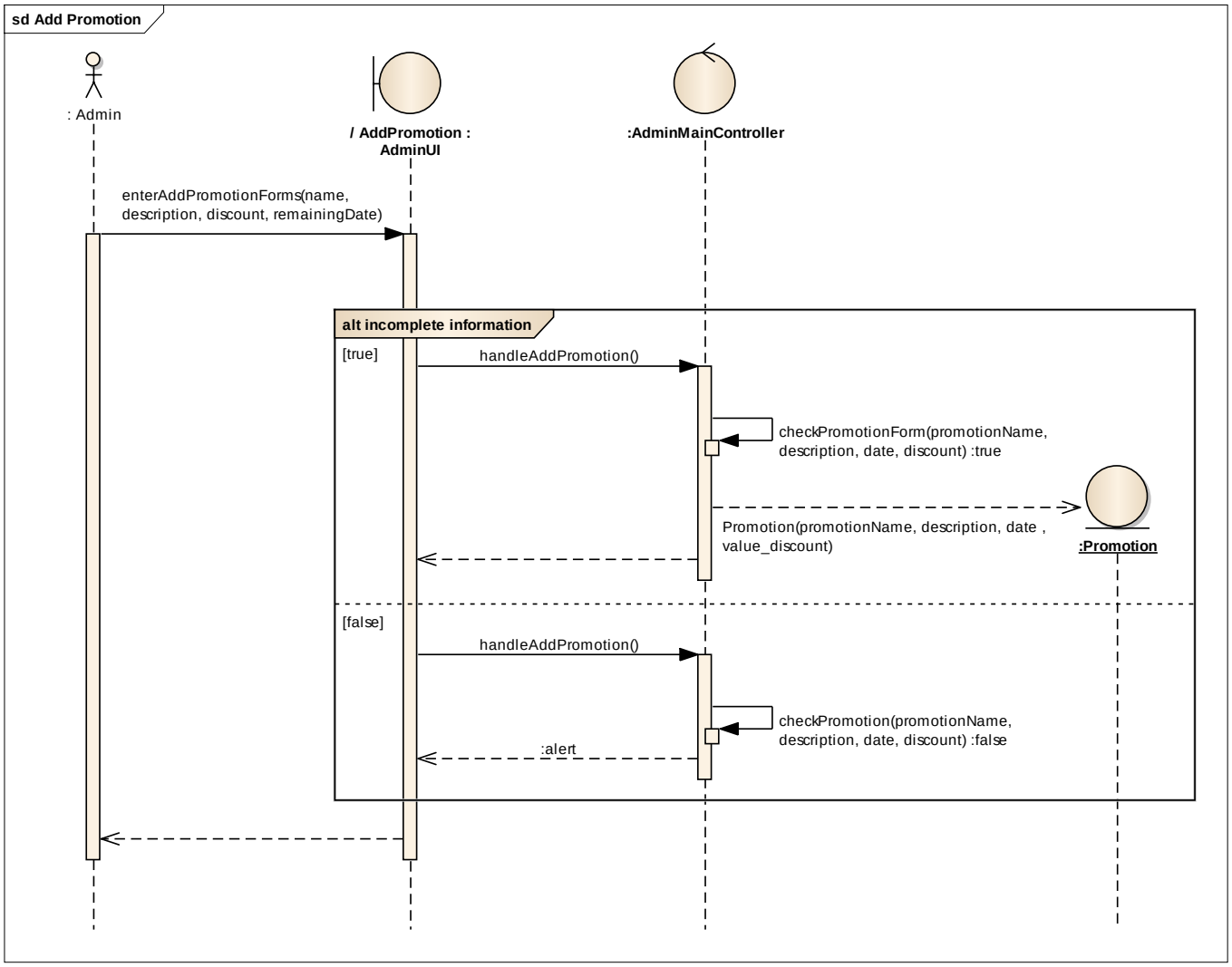
การลบภาพยนตร์



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกภาพยนตร์ ในหน้าเพิ่ม แก้ไข และลบภาพยนตร์ หลังจากนั้นคลิกขวาแล้วเลือก “delete”
2. ระบบจะทำการตรวจสอบภาพยนตร์ที่เลือก และทำการลบ
3. ระบบจะทำการลบภาพยนตร์ และแสดงว่าทำการลบภาพยนตร์เรียบร้อย

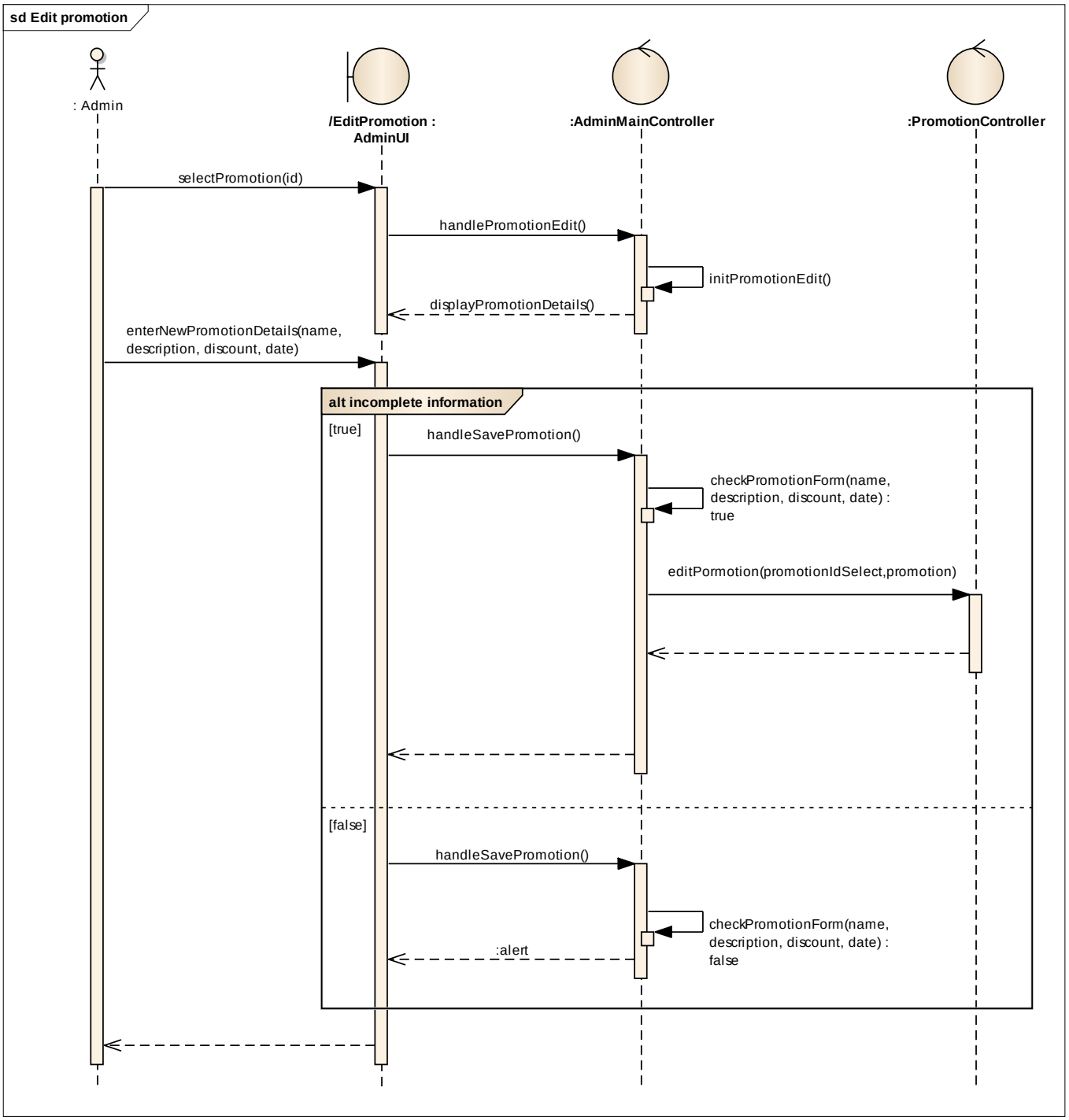
การเพิ่มโปรโมชั่น



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) กรอกข้อมูล ชื่อโปรโมชั่น, คำอธิบาย, ส่วนลด และ วันหมดอายุ ลงในหน้าเพิ่ม แก้ไข และลบโปรโมชั่น หลังจากนั้นกด “Add”
2. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   2.1. ถ้าถูก จะทำงานขั้นต่อไป
   2.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
3. ระบบจะทำการเพิ่มโปรโมชั่น และแสดงว่าทำการเพิ่มโปรโมชั่นเรียบร้อย

การแก้ไขโปรโมชั่น

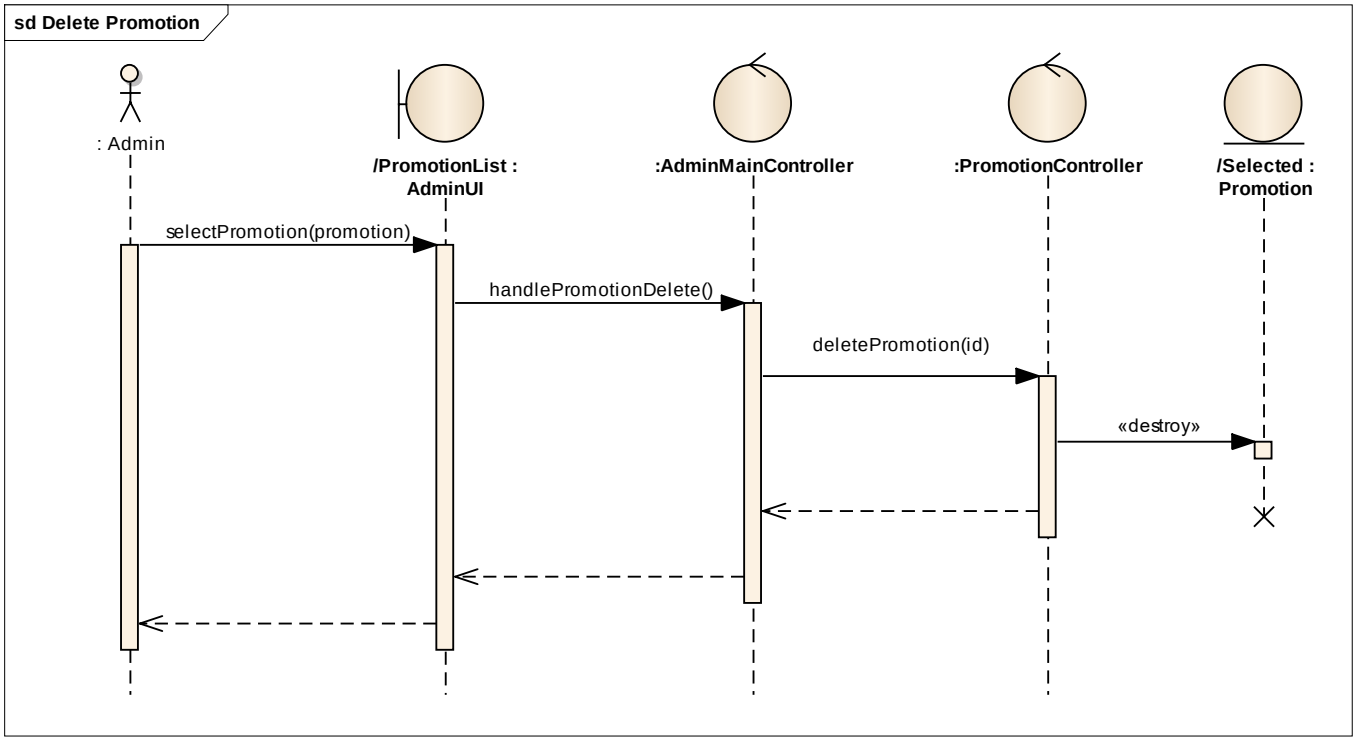


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกโปรโมชั่น
2. ระบบจะทำการตรวจสอบโปรโมชั่นที่เลือก และแสดงรายละเอียดของโปรโมชั่น

3. แก้ไขรายละเอียดซึ่งที่สามารถแก้ไขได้ ได้แก่ ชื่อโปรโมชั่น, คำอธิบาย, ส่วนลด และ วันหมดอายุ ลงในหน้าเพิ่ม แก้ไข และลบโปรโมชั่น หลังจากนั้นกด “Save edit”
4. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   4.1. ถ้าถูก จะทำงานขั้นต่อไป
   4.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
5. ระบบจะทำการแก้ไขข้อมูลโปรโมชั่น และแสดงว่าทำการแก้ไขโปรโมชั่นเรียบร้อย

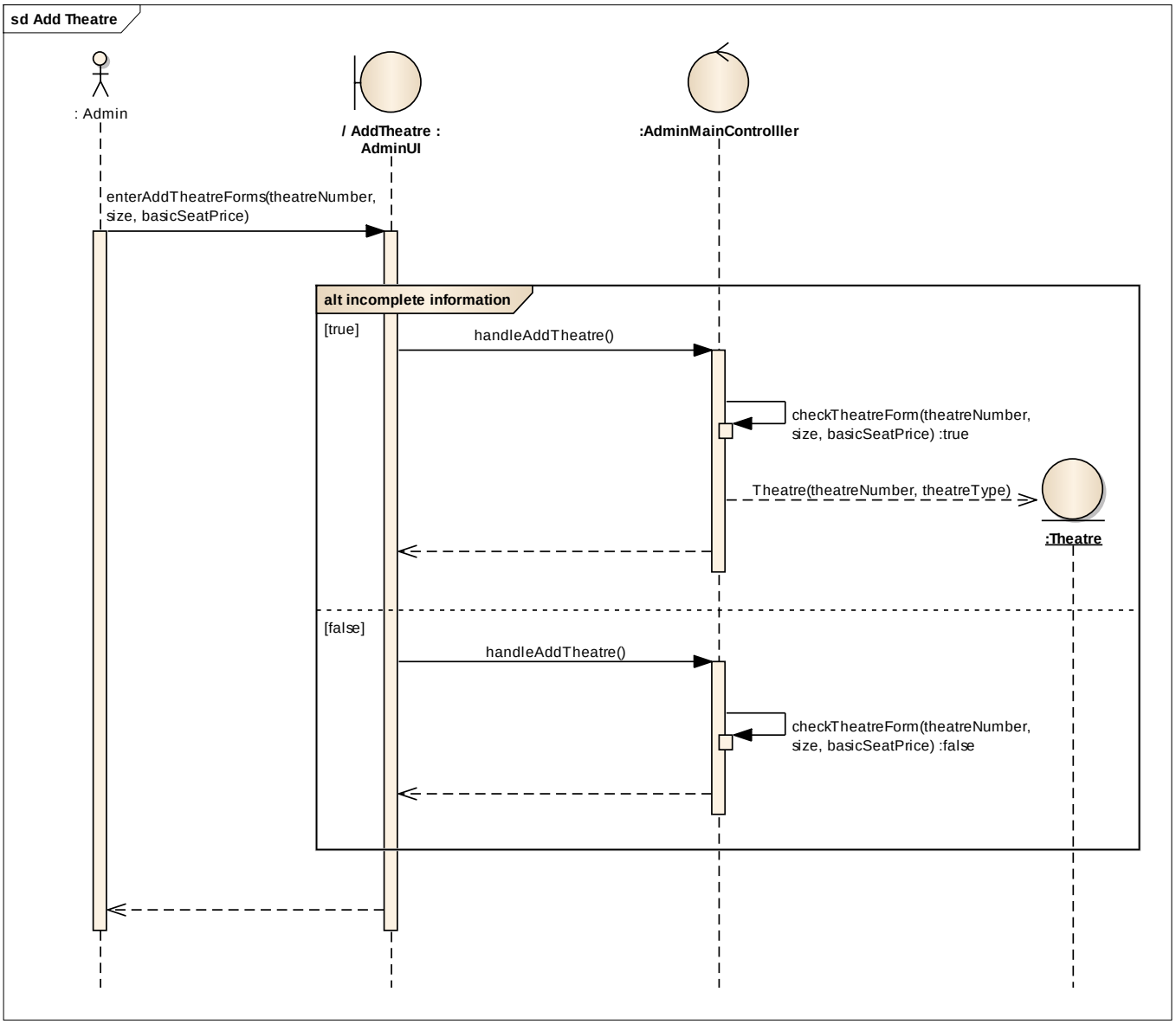
การลบโปรโมชั่น



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกภาพยนตร์ ในหน้าเพิ่ม แก้ไข และลบโปรโมชั่น หลังจากนั้นคลิกขวาแล้วเลือก “delete”
2. ระบบจะทำการตรวจสอบโปรโมชั่นที่เลือก และทำการลบ
3. ระบบจะทำการลบโปรโมชั่น และแสดงว่าทำการลบโปรโมชั่นเรียบร้อย

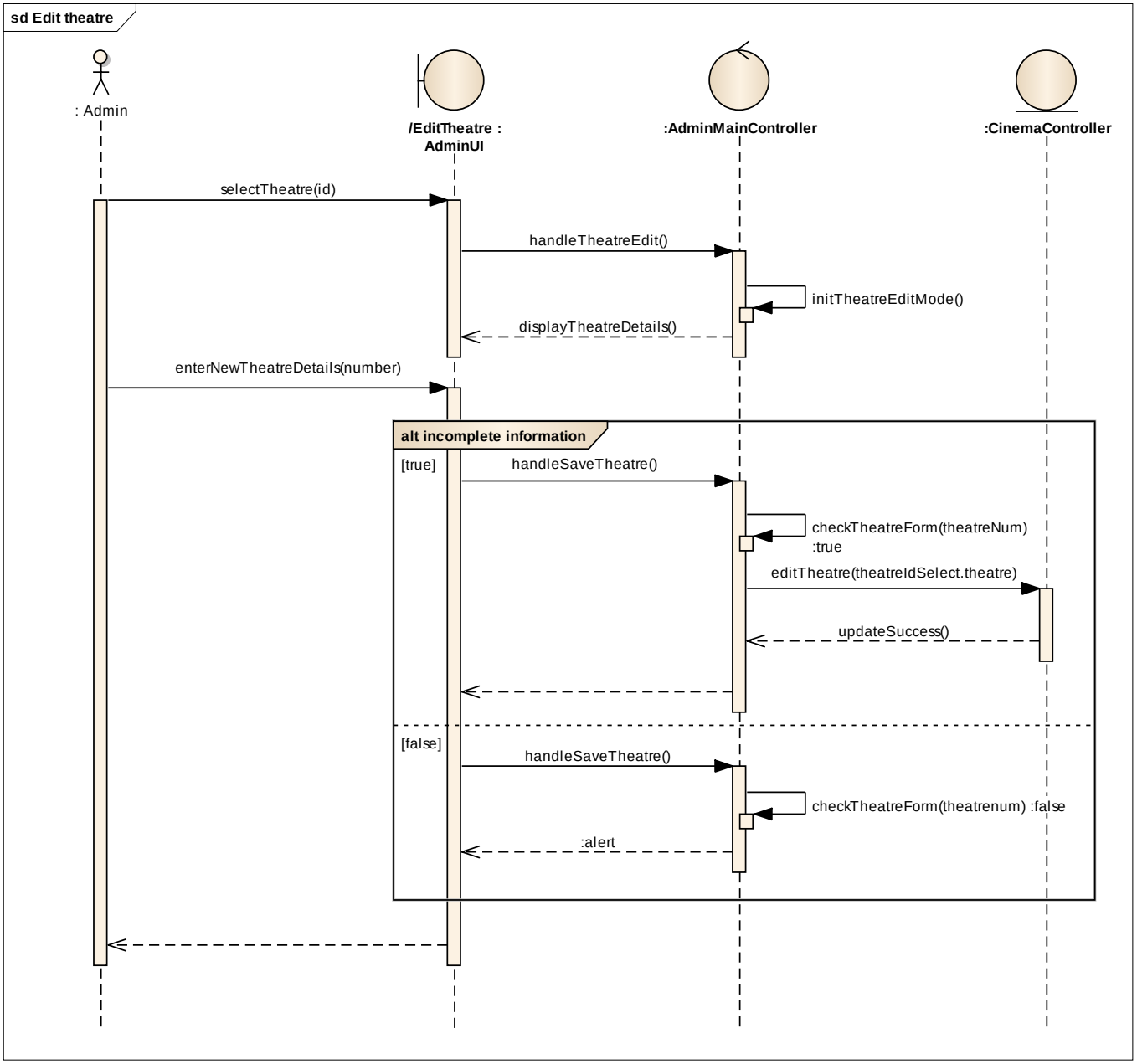
การเพิ่มโรงภาพยนตร์



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) กรอกข้อมูล หมายเลขโรงภาพยนต์, ขนาดของโรงภาพยนต์ และราคาพื้นฐานต่อที่นั่ง ลงในหน้าเพิ่ม แก้ไข และลบโรงภาพยนตร์ หลังจากนั้นกด “Add”
2. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   2.1. ถ้าถูก จะทำงานขั้นต่อไป
   2.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
3. ระบบจะทำการเพิ่มโรงภาพยนตร์ และแสดงว่าทำการเพิ่มโรงภาพยนตร์เรียบร้อย

การแก้ไขโรงภาพยนตร์

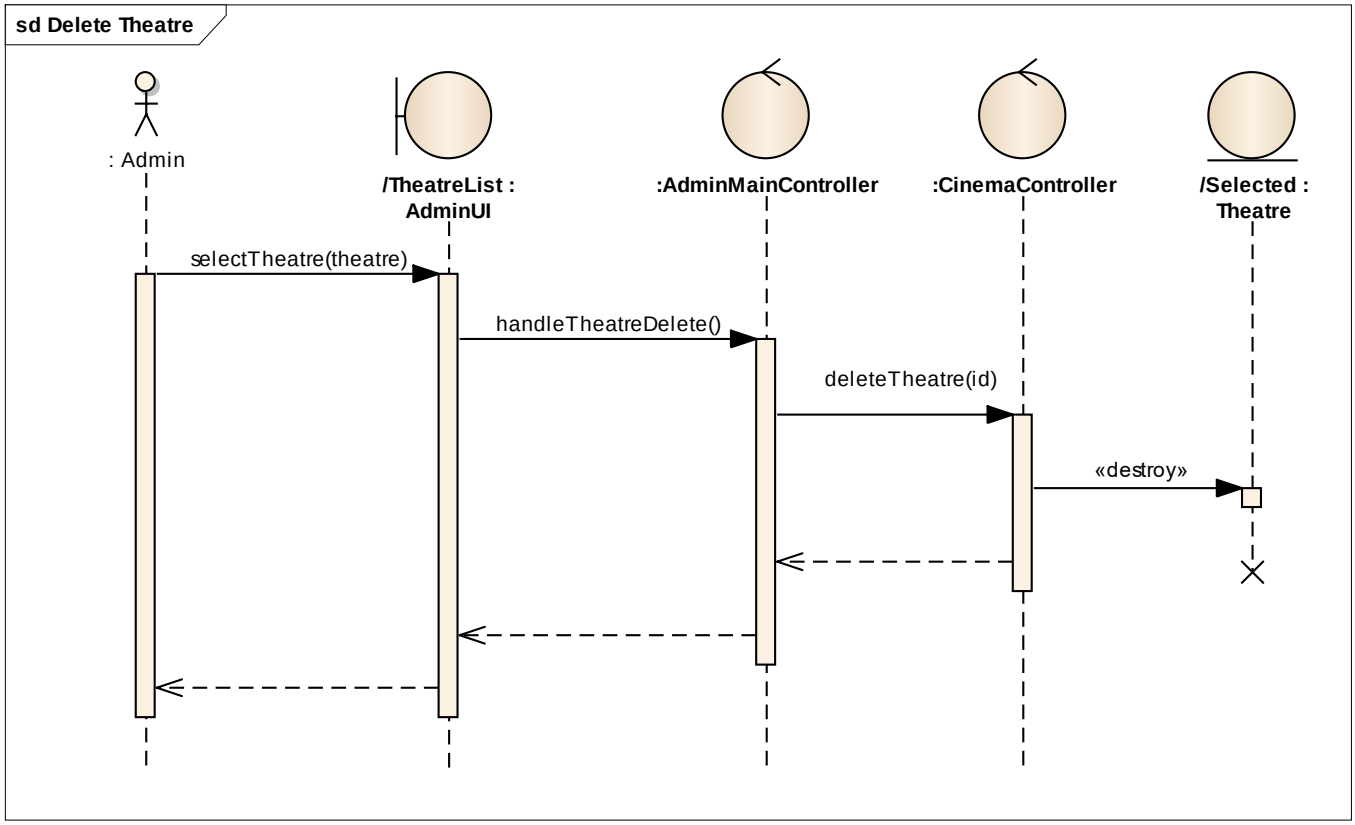


มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกโรงภาพยนตร์
2. ระบบจะทำการตรวจสอบโรงภาพยนตร์ที่เลือก และแสดงรายละเอียดของโรงภาพยนตร์
3. แก้ไขรายละเอียดซึ่งที่สามารถแก้ไขได้ ได้แก่ หมายเลขโรงภาพยนต์, ขนาดของโรงภาพยนต์ และราคาพื้นฐานต่อที่นั่ง ลงในหน้าเพิ่ม แก้ไข และลบโรงภาพยนตร์ หลังจากนั้นกด “Save edit”

4. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   4.1. ถ้าถูก จะทำงานขั้นต่อไป
   4.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
5. ระบบจะทำการแก้ไขข้อมูลโรงภาพยนตร์ และแสดงว่าทำการแก้ไขโรงภาพยนตร์เรียบร้อย

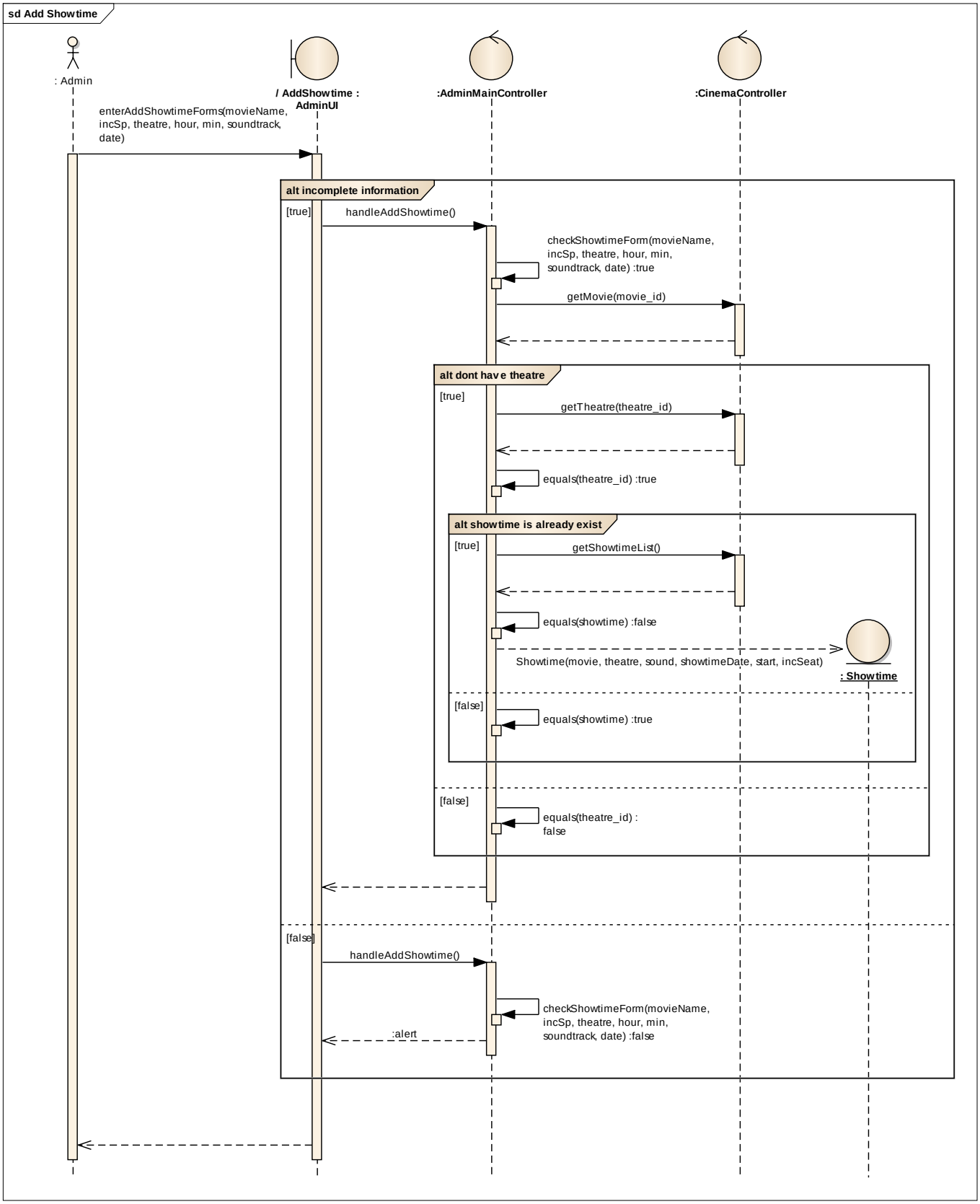
การลบโรงภาพยนตร์



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกภาพยนตร์ ในหน้าเพิ่ม แก้ไข และลบโรงภาพยนตร์ หลังจากนั้นคลิกขวาแล้วเลือก “delete”
2. ระบบจะทำการตรวจสอบโรงภาพยนตร์ที่เลือก และทำการลบ
3. ระบบจะทำการลบโรงภาพยนตร์ และแสดงว่าทำการลบโรงภาพยนตร์เรียบร้อย

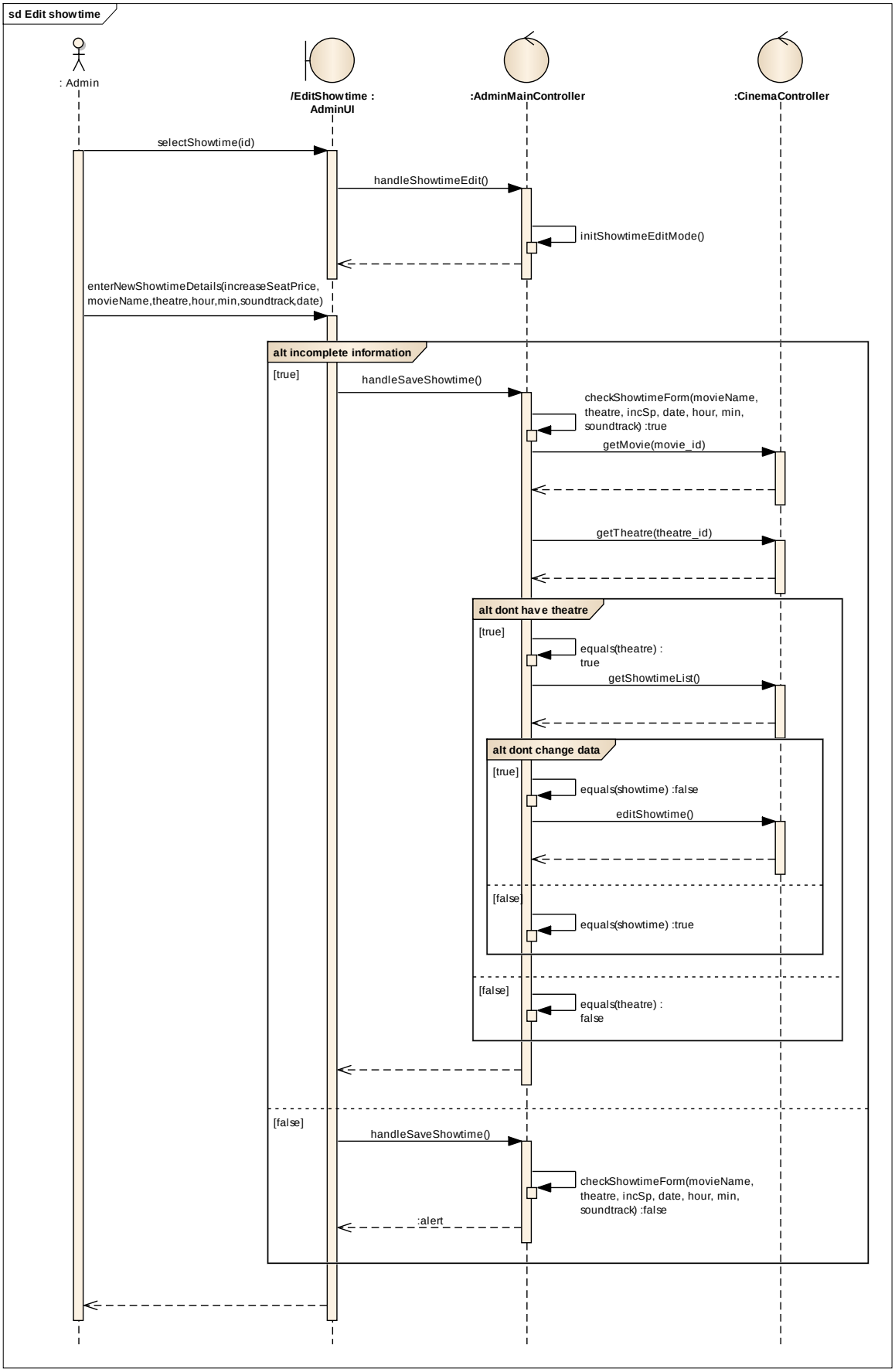
การเพิ่มรอบฉายภาพยนตร์



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) กรอกข้อมูล หมายเลขโรงภาพยนต์, ภาพยนตร์ที่ฉาย, ภาษา, บทบรรยาย, ราคาที่เพิ่มต่อที่นั่ง, เวลา และวันที่ฉาย ลงในหน้าเพิ่ม แก้ไข และลบรอบฉายภาพยนตร์ หลังจากนั้นกด “Add”
2. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   2.1. ถ้าถูก จะทำงานขั้นต่อไป
   2.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
3. ระบบจะทำการตรวจสอบว่ามีโรงภาพยนตร์นี้อยู่ในฐานข้อมูลแล้วหรือไม่
   3.1. ถ้าพบ จะทำงานขั้นต่อไป
   3.2. ถ้าไม่พบ จะแจ้งเตือนว่า ไม่พบโรงภาพยนตร์นี้ ให้ทำการกรอกใหม่
4. ระบบจะทำการตรวจสอบว่ามีรอบฉายภาพยนตร์นี้อยู่ในฐานข้อมูลแล้วหรือไม่
   4.1. ถ้าพบ จะแจ้งเตือนว่า รอบฉายนี้ไม่สามารถใช้ได้ ให้ทำการกรอกใหม่
   4.2. ถ้าไม่พบ จะทำงานขั้นต่อไป
5. ระบบจะทำการเพิ่มรอบฉายภาพยนตร์ และแสดงว่าทำการเพิ่มรอบฉายภาพยนตร์เรียบร้อย

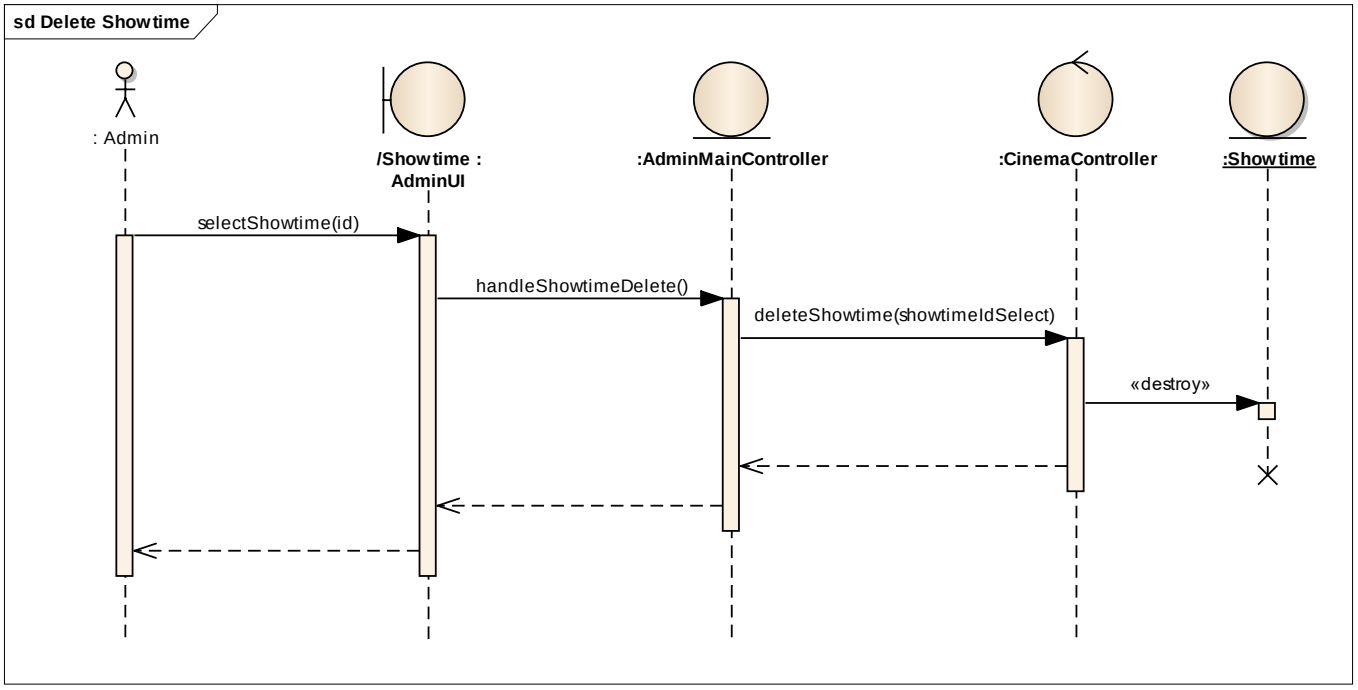
การแก้ไขรอบฉายภาพยนตร์



มีขั้นตอนการทำงานดังนี้

1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกรอบฉายภาพยนตร์
2. ระบบจะทำการตรวจสอบรอบฉายภาพยนตร์ที่เลือก และแสดงรายละเอียดของรอบฉายภาพยนตร์
3. แก้ไขรายละเอียดซึ่งที่สามารถแก้ไขได้ ได้แก่ หมายเลขโรงภาพยนต์, ภาพยนตร์ที่ฉาย, ภาษา, บทบรรยาย, ราคาที่เพิ่มต่อที่นั่ง, เวลา และวันที่ฉาย ลงในหน้าเพิ่ม แก้ไข และลบรอบฉายภาพยนตร์ หลังจากนั้นกด “Save edit”
4. ระบบจะทำการตรวจสอบว่ากรอกรายละเอียดถูกรูปแบบหรือไม่
   4.1. ถ้าถูก จะทำงานขั้นต่อไป
   4.2. ถ้าผิด จะทำการแจ้งเตือนว่า กรอกข้อมูลไม่ถูกต้อง ให้ทำการกรอกใหม่
5. ระบบจะทำการตรวจสอบว่ามีโรงภาพยนตร์นี้อยู่ในฐานข้อมูลแล้วหรือไม่
   5.1. ถ้าพบ จะทำงานขั้นต่อไป
   5.2. ถ้าไม่พบ จะแจ้งเตือนว่า ไม่พบโรงภาพยนตร์นี้ ให้ทำการกรอกใหม่
6. ระบบจะทำการตรวจสอบว่ามีการแก้ไขข้อมูลหรือไม่
   6.1. ถ้าแก้ไข จะทำงานขั้นต่อไป
   6.2. ถ้าไม่แก้ไข จะไม่ทำการแก้ไขข้อมูล
7. ระบบจะทำการแก้ไขรอบฉายภาพยนตร์ และแสดงว่าทำการแก้ไขรอบฉายภาพยนตร์เรียบร้อย

การลบรอบฉายภาพยนตร์



มีขั้นตอนการทำงานดังนี้

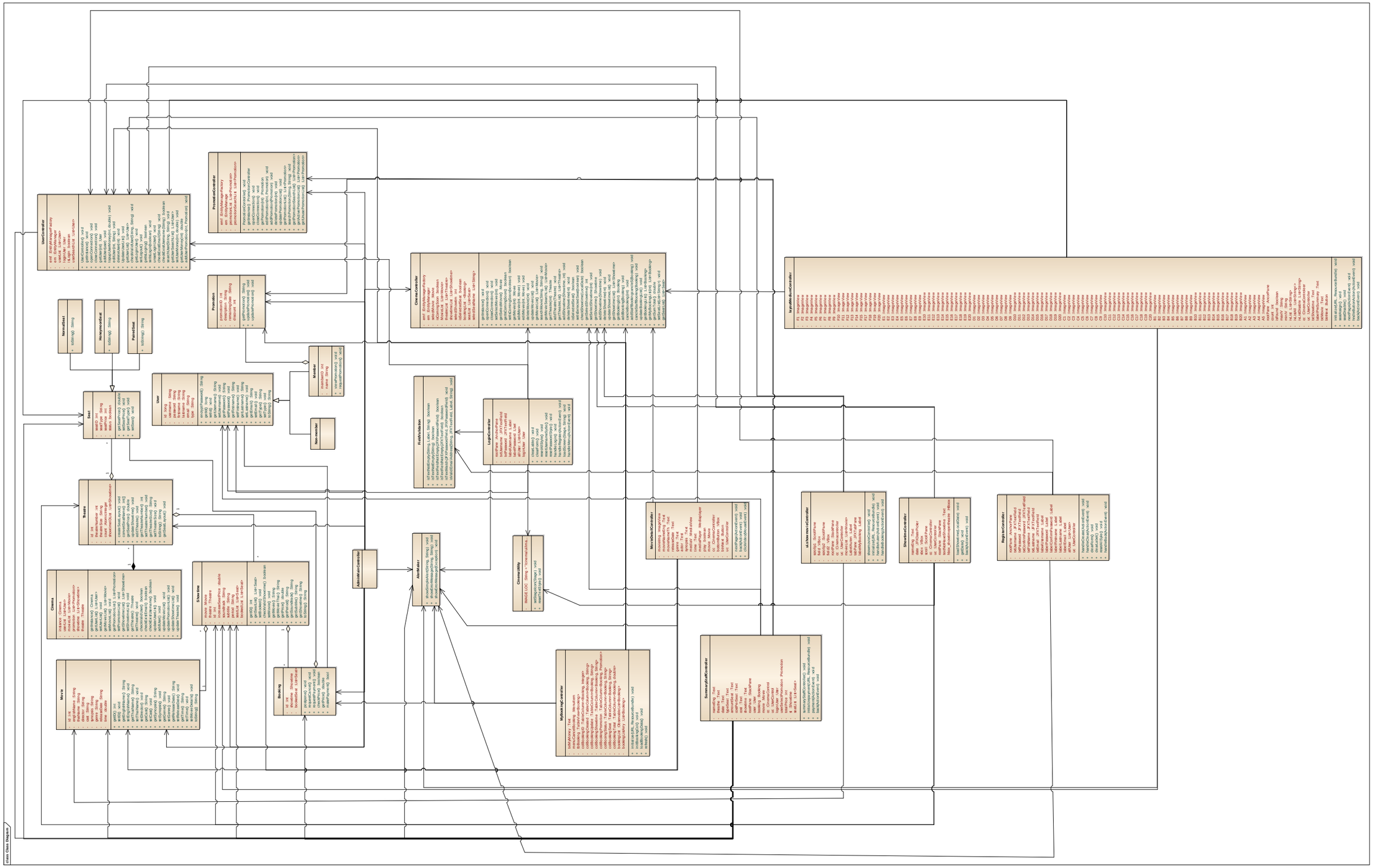
1. ผู้ใช้งานที่มีบทบาทเป็นผู้ดูแลระบบ (Admin) เลือกรอบฉายภาพยนตร์ ในหน้าเพิ่ม แก้ไข และลบรอบฉายภาพยนตร์ หลังจากนั้นคลิกขวาแล้วเลือก “delete”
2. ระบบจะทำการตรวจสอบรอบฉายภาพยนตร์ที่เลือก และทำการลบ
3. ระบบจะทำการลบรอบฉายภาพยนตร์ และแสดงว่าทำการลบรอบฉายภาพยนตร์เรียบร้อย

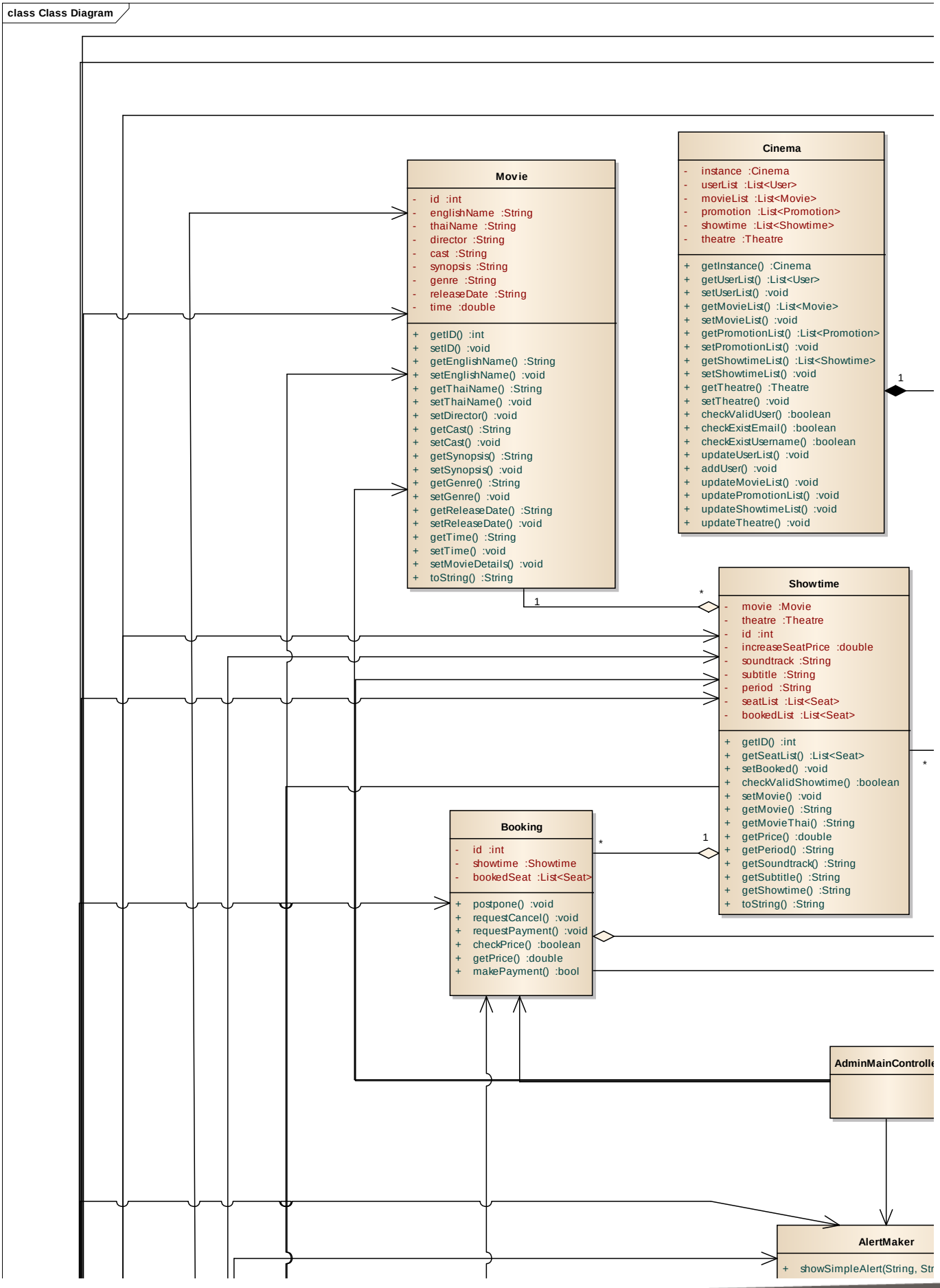
Chapter 3

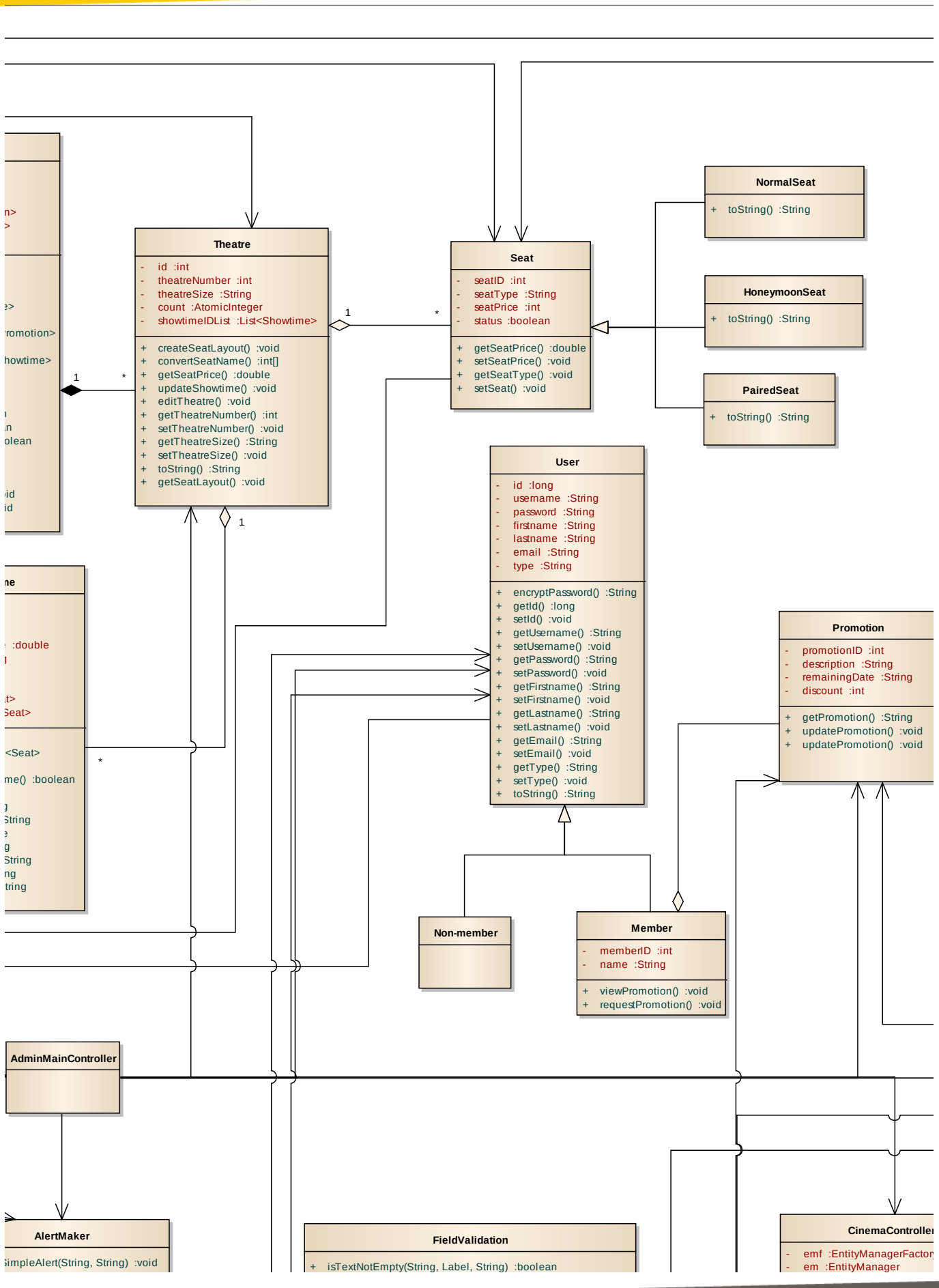
Design

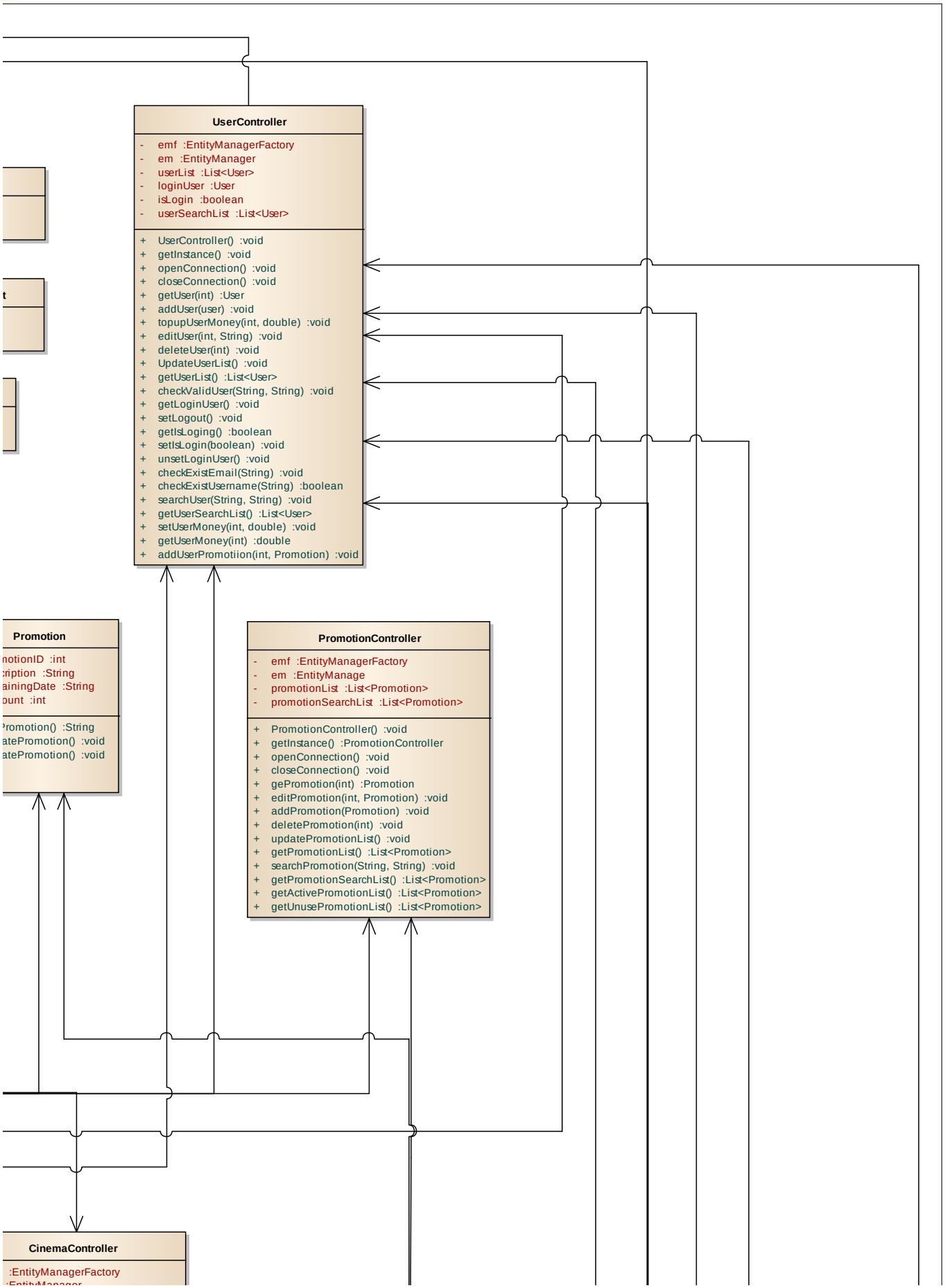
KMITL Cinema

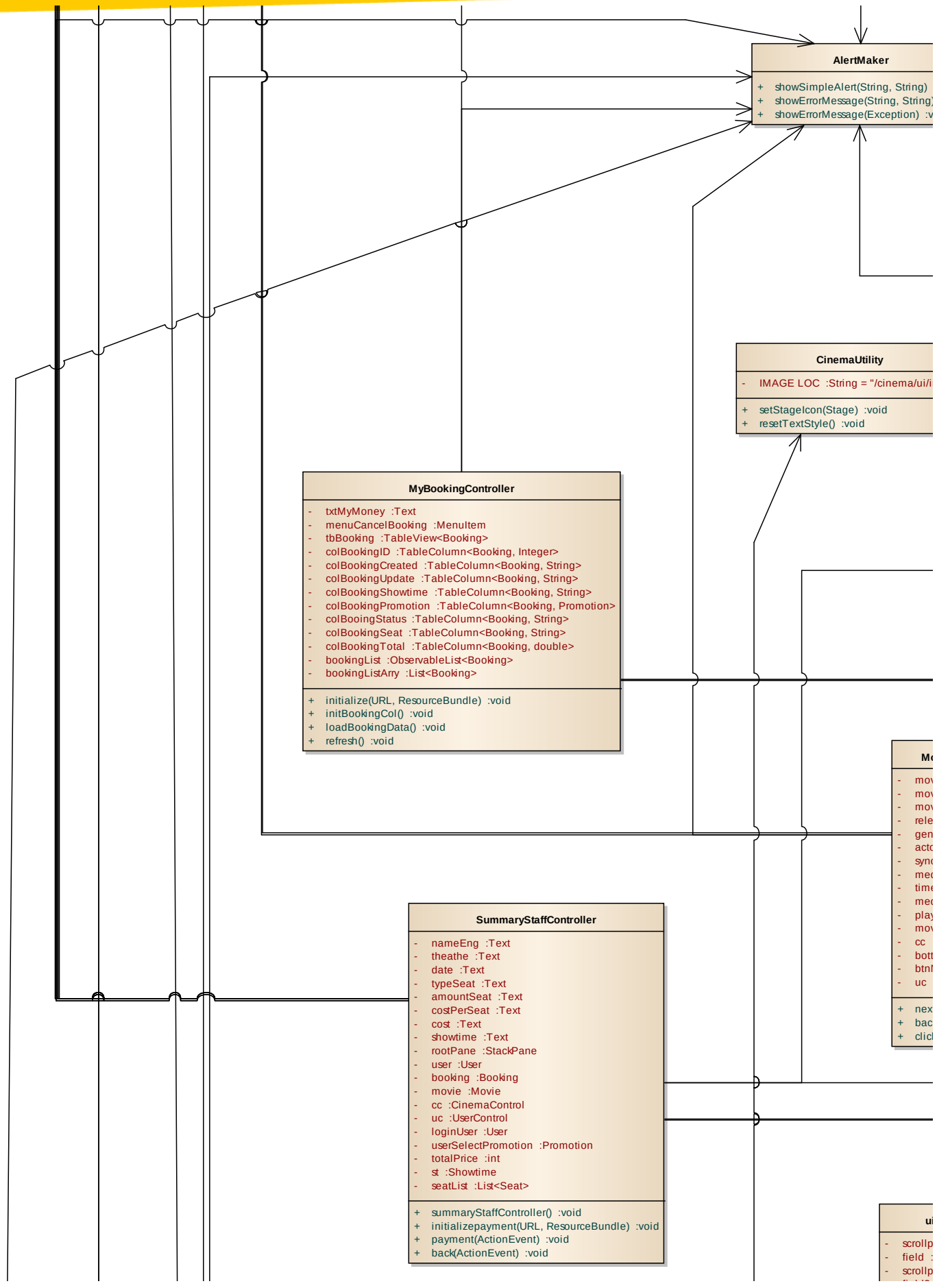
Class diagram

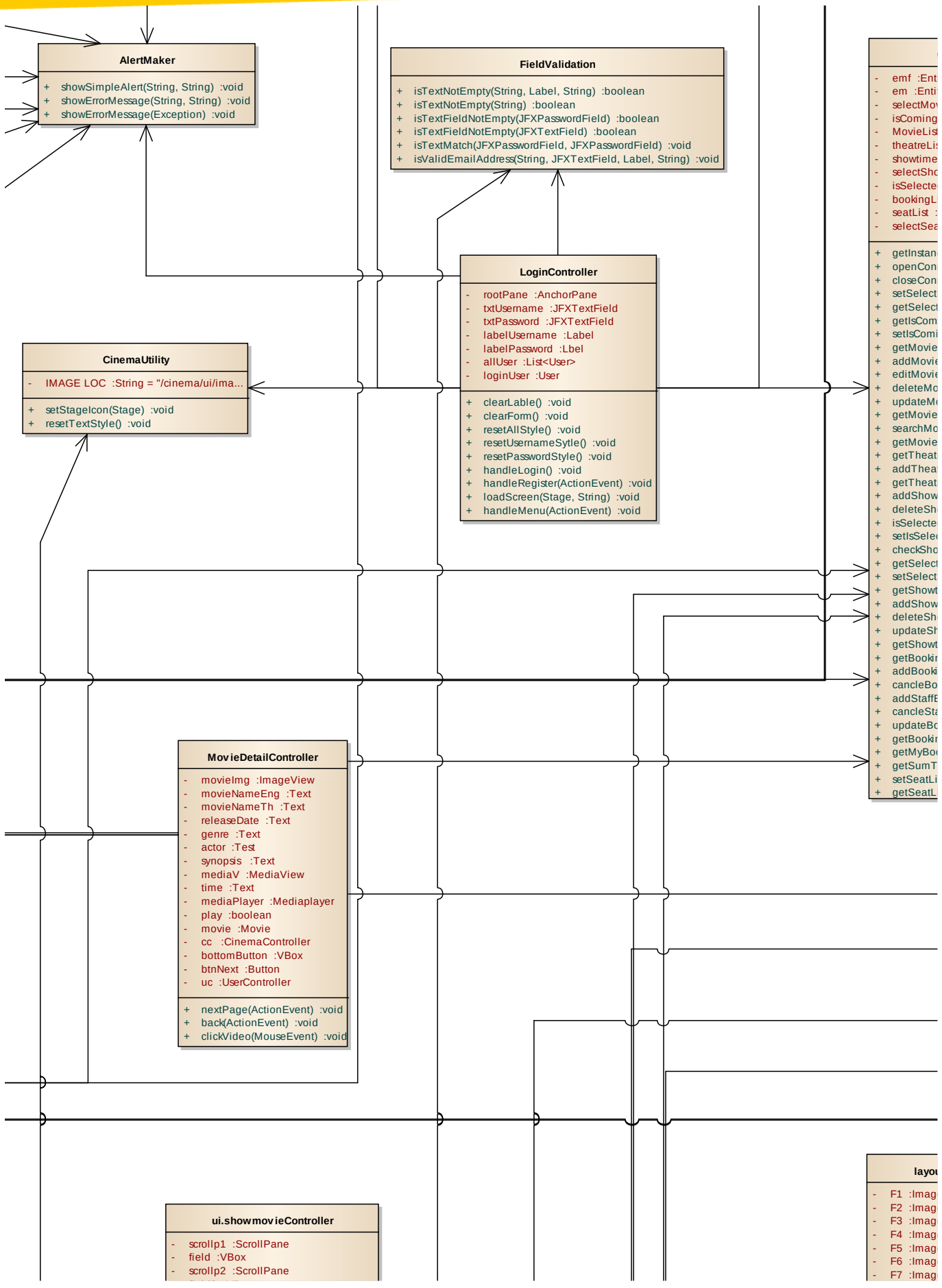


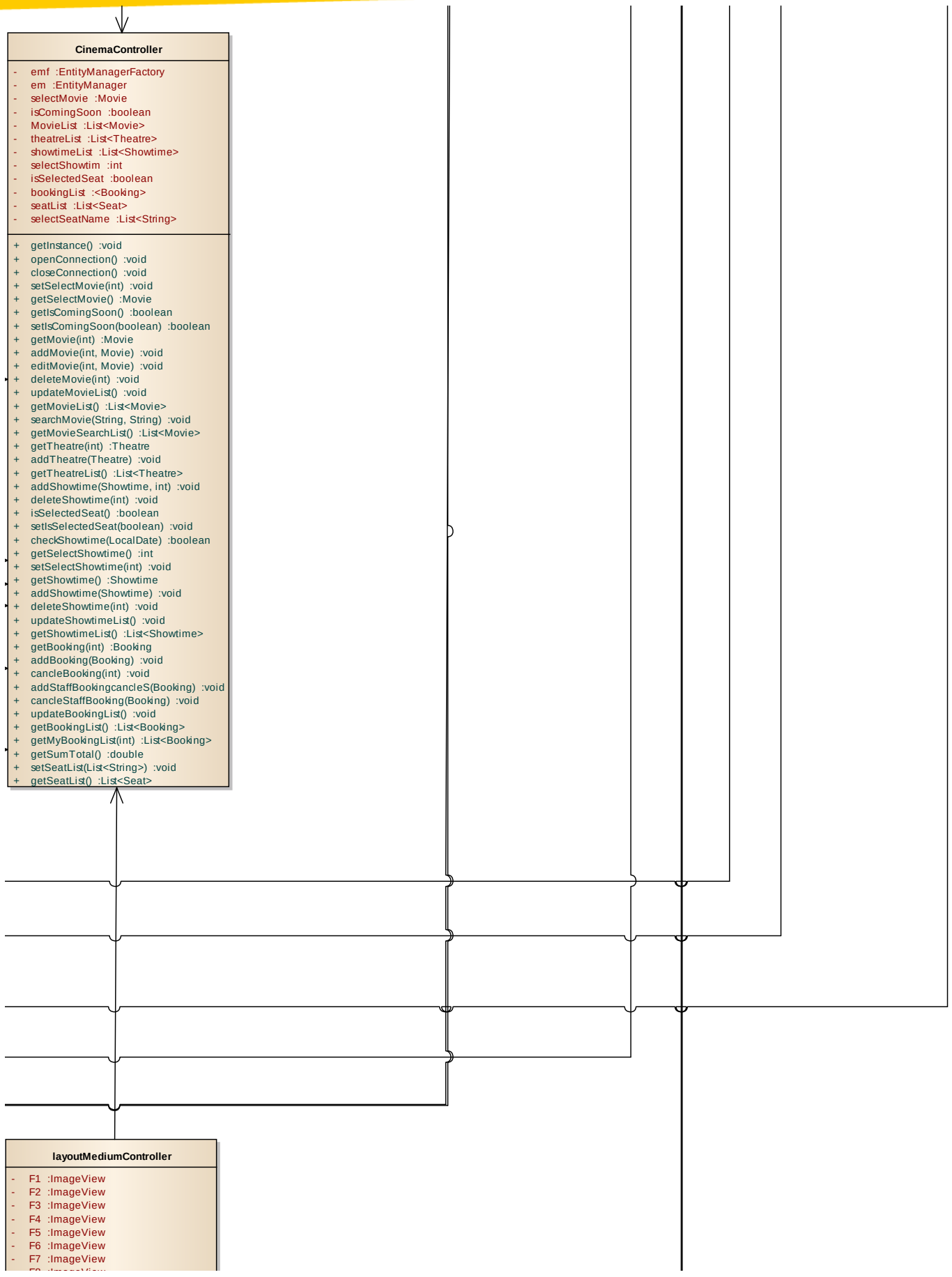

class Class Diagram
Movie
- id :int
- englishName :String
- thaiName :String
- director :String
- cast :String
- synopsis :String
- genre :String
- releaseDate :String
- time :double
+ getID() :int
+ setID() :void
+ getEnglishName() :String
+ setEnglishName() :void
+ getThaiName() :String
+ setThaiName() :void
+ setDirector() :void
+ getCast() :String
+ setCast() :void
+ getSynopsis() :String
+ setSynopsis() :void
+ getGenre() :String
+ setGenre() :void
+ getReleaseDate() :String
+ setReleaseDate() :void
+ getTime() :String
+ setTime() :void
+ setMovieDetails() :void
+ toString() :String
Cinema
- instance :Cinema
- userList :List<User>
- movieList :List<Movie>
- promotion :List<Promotion>
- showtime :List<Showtime>
- theatre :Theatre
+ getInstance() :Cinema
+ getUserList() :List<User>
+ setUserList() :void
+ getMovieList() :List<Movie>
+ setMovieList() :void
+ getPromotionList() :List<Promotion>
+ setPromotionList() :void
+ getShowtimeList() :List<Showtime>
+ setShowtimeList() :void
+ getTheatre() :Theatre
+ setTheatre() :void
+ checkValidUser() :boolean
+ checkExistEmail() :boolean
+ checkExistUsername() :boolean
+ updateUserList() :void
+ addUser() :void
+ updateMovieList() :void
+ updatePromotionList() :void
+ updateShowtimeList() :void
+ updateTheatre() :void
Showtime
- movie :Movie
- theatre :Theatre
- id :int
- increaseSeatPrice :double
- soundtrack :String
- subtitle :String
- period :String
- seatList :List<Seat>
- bookedList :List<Seat>
+ getID() :int
+ getSeatList() :List<Seat>
+ setBooked() :void
+ checkValidShowtime() :boolean
+ setMovie() :void
+ getMovie() :String
+ getMovieThai() :String
+ getPrice() :double
+ getPeriod() :String
+ getSoundtrack() :String
+ getSubtitle() :String
+ getShowtime() :String
+ toString() :String
Booking
- id :int
- showtime :Showtime
- bookedSeat :List<Seat>
+ postpone() :void
+ requestCancel() :void
+ requestPayment() :void
+ checkPrice() :boolean
+ getPrice() :double
+ makePayment() :bool
AdminMainControlle
AlertMaker
+ showSimpleAlert(String, Str
1
*
1
*
1
*

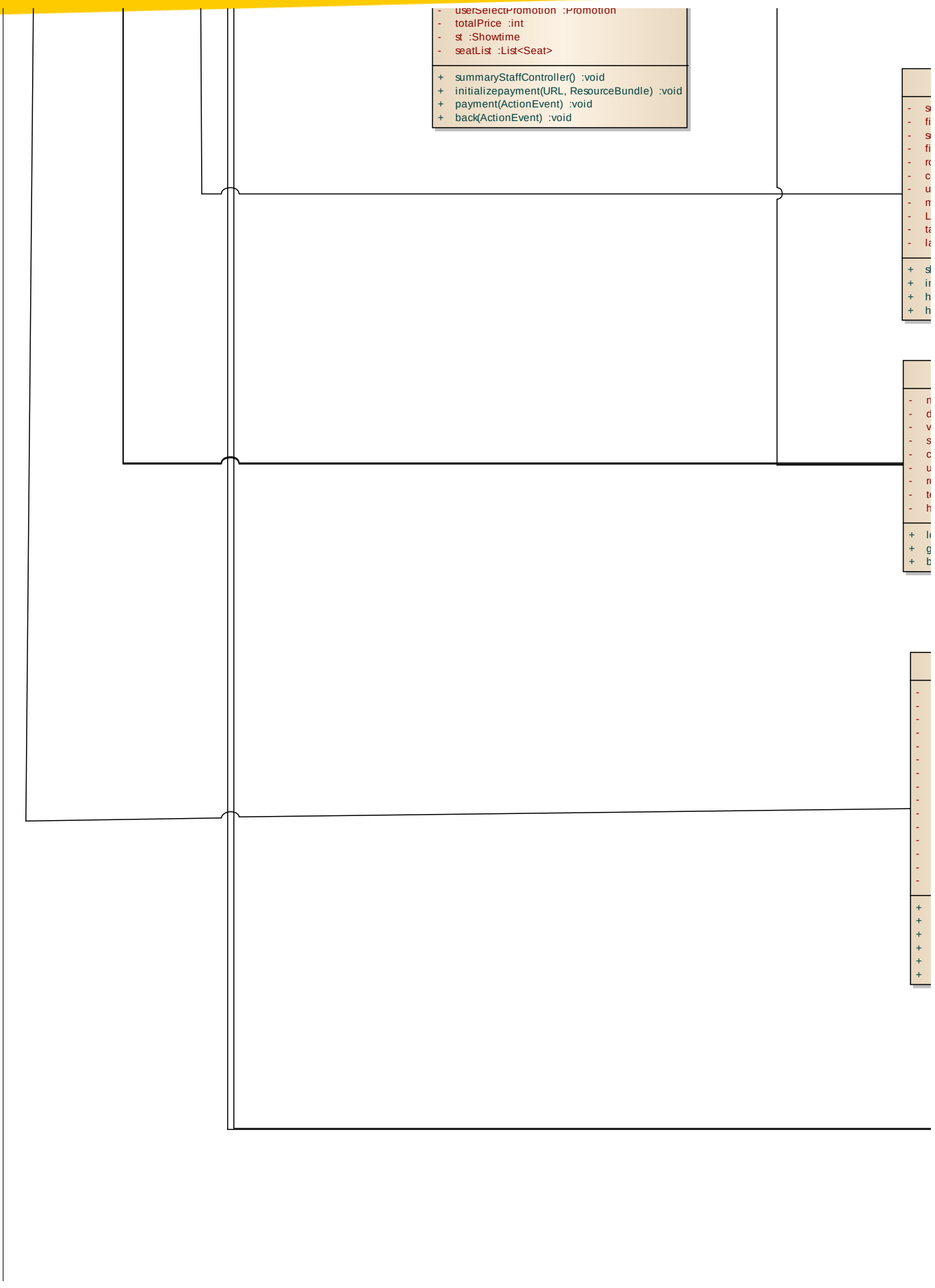

NormalSeat
+ toString() :String
Theatre
- id :int
- theatreNumber :int
- theatreSize :String
- count :AtomicInteger
- showtimeIDList :List<Showtime>
+ createSeatLayout() :void
+ convertSeatName() :int[]
+ getSeatPrice() :double
+ updateShowtime() :void
+ editTheatre() :void
+ getTheatreNumber() :int
+ setTheatreNumber() :void
+ getTheatreSize() :String
+ setTheatreSize() :void
+ toString() :String
+ getSeatLayout() :void
Seat
- seatID :int
- seatType :String
- seatPrice :int
- status :boolean
+ getSeatPrice() :double
+ setSeatPrice() :void
+ getSeatType() :void
+ setSeat() :void
HoneymoonSeat
+ toString() :String
PairedSeat
+ toString() :String
User
- id :long
- username :String
- password :String
- firstname :String
- lastname :String
- email :String
- type :String
+ encryptPassword() :String
+ getId() :long
+ setId() :void
+ getUsername() :String
+ setUsername() :void
+ getPassword() :String
+ setPassword() :void
+ getFirstname() :String
+ setFirstname() :void
+ getLastname() :String
+ setLastname() :void
+ getEmail() :String
+ setEmail() :void
+ getType() :String
+ setType() :void
+ toString() :String
Promotion
- promotionID :int
- description :String
- remainingDate :String
- discount :int
+ getPromotion() :String
+ updatePromotion() :void
+ updatePromotion() :void
Non-member
Member
- memberID :int
- name :String
+ viewPromotion() :void
+ requestPromotion() :void
AdminMainController
AlertMaker
FieldValidation
+ isTextNotEmpty(String, Label, String) :boolean
1
*

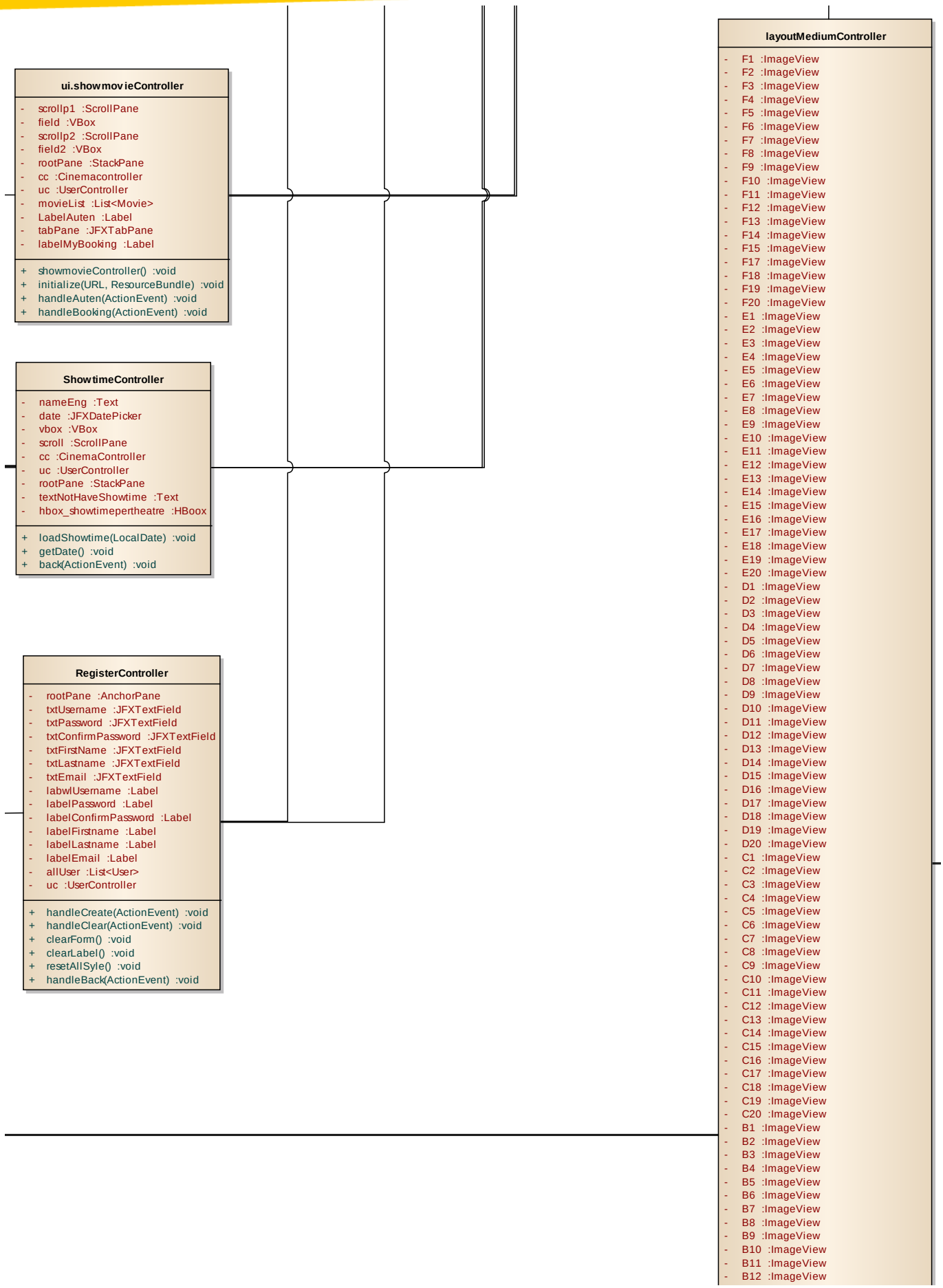

UserController
- emf :EntityManagerFactory
- em :EntityManager
- userList :List<User>
- loginUser :User
- isLogin :boolean
- userSearchList :List<User>
+ UserController() :void
+ getInstance() :void
+ openConnection() :void
+ closeConnection() :void
+ getUser(int) :User
+ addUser(user) :void
+ topupUserMoney(int, double) :void
+ editUser(int, String) :void
+ deleteUser(int) :void
+ UpdateUserList() :void
+ getUserList() :List<User>
+ checkValidUser(String, String) :void
+ getLoginUser() :void
+ setLogout() :void
+ getIsLoging() :boolean
+ setIsLogin(boolean) :void
+ unsetLoginUser() :void
+ checkExistEmail(String) :void
+ checkExistUsername(String) :boolean
+ searchUser(String, String) :void
+ getUserSearchList() :List<User>
+ setUserMoney(int, double) :void
+ getUserMoney(int) :double
+ addUserPromotiion(int, Promotion) :void
Promotion
PromotionController
- emf :EntityManagerFactory
- em :EntityManage
- promotionList :List<Promotion>
- promotionSearchList :List<Promotion>
+ PromotionController() :void
+ getInstance() :PromotionController
+ openConnection() :void
+ closeConnection() :void
+ gePromotion(int) :Promotion
+ editPromotion(int, Promotion) :void
+ addPromotion(Promotion) :void
+ deletePromotion(int) :void
+ updatePromotionList() :void
+ getPromotionList() :List<Promotion>
+ searchPromotion(String, String) :void
+ getPromotionSearchList() :List<Promotion>
+ getActivePromotionList() :List<Promotion>
+ getUnusePromotionList() :List<Promotion>
CinemaController
:EntityManagerFactory

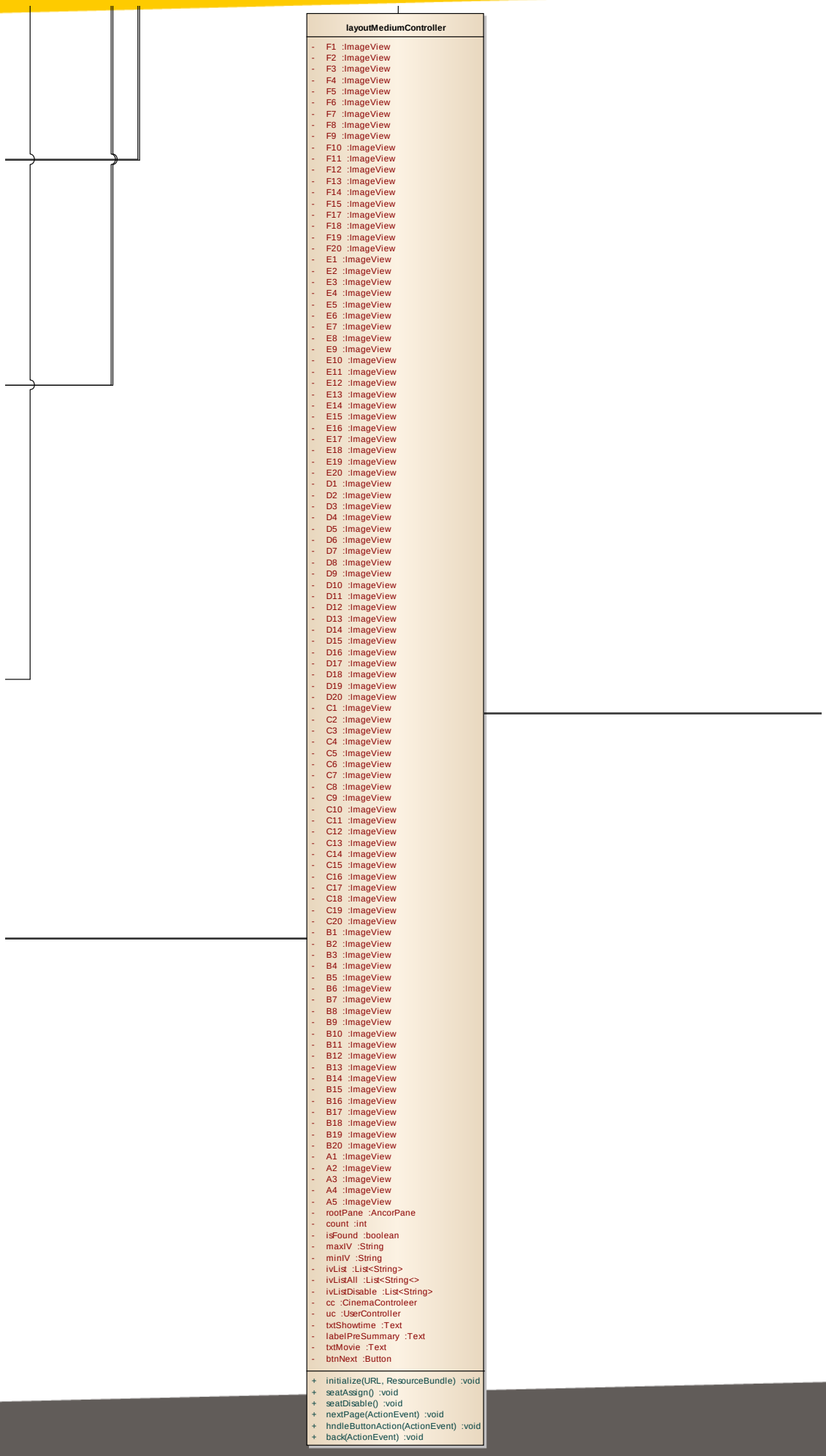

AlertMaker
+ showSimpleAlert(String, String)
+ showErrorMessage(String, String)
+ showErrorMessage(Exception) :v
CinemaUtility
- IMAGE LOC :String = "/cinema/ui/i
+ setStageIcon(Stage) :void
+ resetTextStyle() :void
MyBookingController
- txtMyMoney :Text
- menuCancelBooking :MenuItem
- tbBooking :TableView<Booking>
- colBookingID :TableColumn<Booking, Integer>
- colBookingCreated :TableColumn<Booking, String>
- colBookingUpdate :TableColumn<Booking, String>
- colBookingShowtime :TableColumn<Booking, String>
- colBookingPromotion :TableColumn<Booking, Promotion>
- colBooingStatus :TableColumn<Booking, String>
- colBookingSeat :TableColumn<Booking, String>
- colBookingTotal :TableColumn<Booking, double>
- bookingList :ObservableList<Booking>
- bookingListArry :List<Booking>
+ initialize(URL, ResourceBundle) :void
+ initBookingCol() :void
+ loadBookingData() :void
+ refresh() :void
SummaryStaffController
- nameEng :Text
- theathe :Text
- date :Text
- typeSeat :Text
- amountSeat :Text
- costPerSeat :Text
- cost :Text
- showtime :Text
- rootPane :StackPane
- user :User
- booking :Booking
- movie :Movie
- cc :CinemaControl
- uc :UserControl
- loginUser :User
- userSelectPromotion :Promotion
- totalPrice :int
- st :Showtime
- seatList :List<Seat>
+ summaryStaffController() :void
+ initializepayment(URL, ResourceBundle) :void
+ payment(ActionEvent) :void
+ back(ActionEvent) :void


AlertMaker
+ showSimpleAlert(String, String) :void
+ showErrorMessage(String, String) :void
+ showErrorMessage(Exception) :void
FieldValidation
+ isTextNotEmpty(String, Label, String) :boolean
+ isTextNotEmpty(String) :boolean
+ isTextFieldNotEmpty(JFXPasswordField) :boolean
+ isTextFieldNotEmpty(JFXTextField) :boolean
+ isTextMatch(JFXPasswordField, JFXPasswordField) :void
+ isValidEmailAddress(String, JFXTextField, Label, String) :void
LoginController
- rootPane :AnchorPane
- txtUsername :JFXTextField
- txtPassword :JFXTextField
- labelUsername :Label
- labelPassword :Lbel
- allUser :List<User>
- loginUser :User
+ clearLable() :void
+ clearForm() :void
+ resetAllStyle() :void
+ resetUsernameSytle() :void
+ resetPasswordStyle() :void
+ handleLogin() :void
+ handleRegister(ActionEvent) :void
+ loadScreen(Stage, String) :void
+ handleMenu(ActionEvent) :void
CinemaUtility
- IMAGE LOC :String = "/cinema/ui/ima...
+ setStageIcon(Stage) :void
+ resetTextStyle() :void
MovieDetailController
- movieImg :ImageView
- movieNameEng :Text
- movieNameTh :Text
- releaseDate :Text
- genre :Text
- actor :Test
- synopsis :Text
- mediaV :MediaView
- time :Text
- mediaPlayer :Mediaplayer
- play :boolean
- movie :Movie
- cc :CinemaController
- bottomButton :VBox
- btnNext :Button
- uc :UserController
+ nextPage(ActionEvent) :void
+ back(ActionEvent) :void
+ clickVideo(MouseEvent) :void
ui.showmovieController
- scrollp1 :ScrollPane
- field :VBox
- scrollp2 :ScrollPane


CinemaController
- emf :EntityManagerFactory
- em :EntityManager
- selectMovie :Movie
- isComingSoon :boolean
- MovieList :List<Movie>
- theatreList :List<Theatre>
- showtimeList :List<Showtime>
- selectShowtim :int
- isSelectedSeat :boolean
- bookingList :<Booking>
- seatList :List<Seat>
- selectSeatName :List<String>
+ getInstance() :void
+ openConnection() :void
+ closeConnection() :void
+ setSelectMovie(int) :void
+ getSelectMovie() :Movie
+ getIsComingSoon() :boolean
+ setIsComingSoon(boolean) :boolean
+ getMovie(int) :Movie
+ addMovie(int, Movie) :void
+ editMovie(int, Movie) :void
+ deleteMovie(int) :void
+ updateMovieList() :void
+ getMovieList() :List<Movie>
+ searchMovie(String, String) :void
+ getMovieSearchList() :List<Movie>
+ getTheatre(int) :Theatre
+ addTheatre(Theatre) :void
+ getTheatreList() :List<Theatre>
+ addShowtime(Showtime, int) :void
+ deleteShowtime(int) :void
+ isSelectedSeat() :boolean
+ setIsSelectedSeat(boolean) :void
+ checkShowtime(LocalDate) :boolean
+ getSelectShowtime() :int
+ setSelectShowtime(int) :void
+ getShowtime() :Showtime
+ addShowtime(Showtime) :void
+ deleteShowtime(int) :void
+ updateShowtimeList() :void
+ getShowtimeList() :List<Showtime>
+ getBooking(int) :Booking
+ addBooking(Booking) :void
+ cancleBooking(int) :void
+ addStaffBookingcancleS(Booking) :void
+ cancleStaffBooking(Booking) :void
+ updateBookingList() :void
+ getBookingList() :List<Booking>
+ getMyBookingList(int) :List<Booking>
+ getSumTotal() :double
+ setSeatList(List<String>) :void
+ getSeatList() :List<Seat>
layoutMediumController
- F1 :ImageView
- F2 :ImageView
- F3 :ImageView
- F4 :ImageView
- F5 :ImageView
- F6 :ImageView
- F7 :ImageView

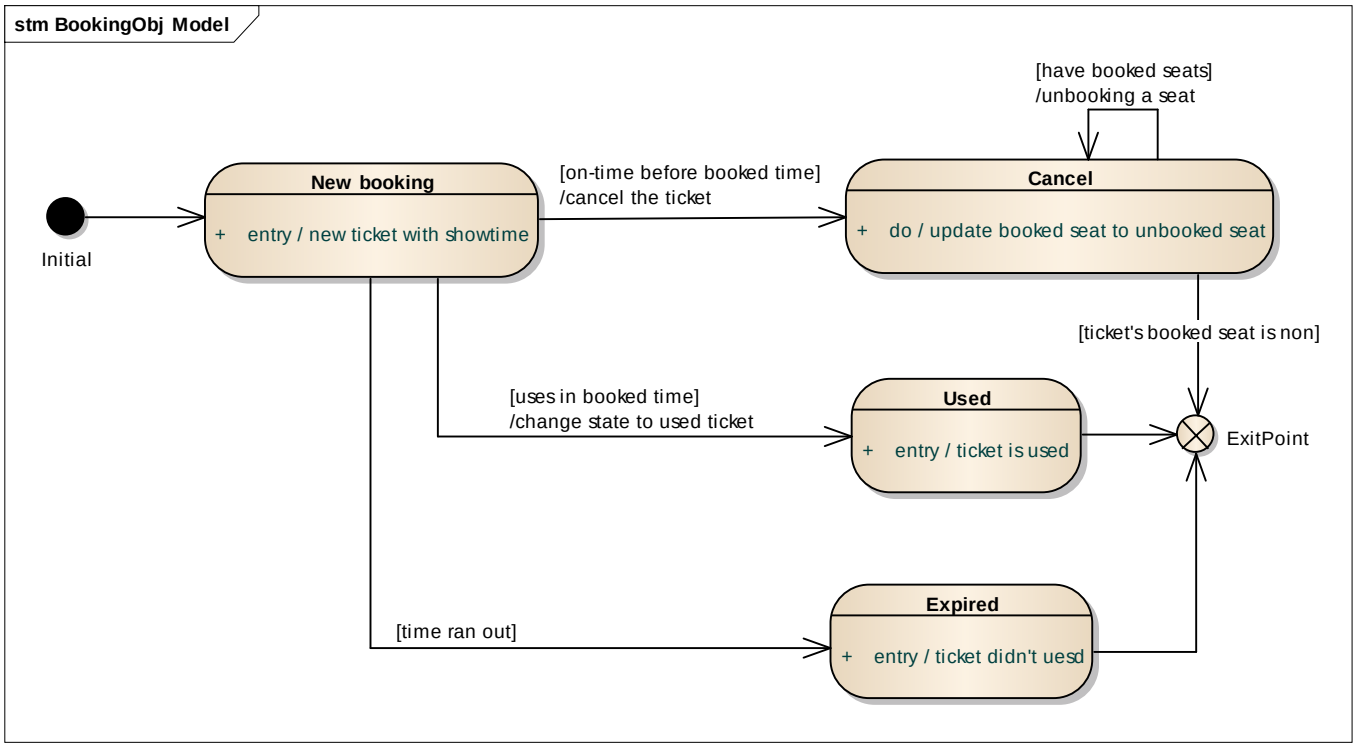

- userSelectPromotion :Promotion
- totalPrice :int
- st :Showtime
- seatList :List<Seat>
+ summaryStaffController() :void
+ initializepayment(URL, ResourceBundle) :void
+ payment(ActionEvent) :void
+ back(ActionEvent) :void


ui.showmovieController
- scrollp1 :ScrollPane
- field :VBox
- scrollp2 :ScrollPane
- field2 :VBox
- rootPane :StackPane
- cc :Cinemacontroller
- uc :UserController
- movieList :List<Movie>
- LabelAuten :Label
- tabPane :JFXTabPane
- labelMyBooking :Label
+ showmovieController() :void
+ initialize(URL, ResourceBundle) :void
+ handleAuten(ActionEvent) :void
+ handleBooking(ActionEvent) :void
ShowtimeController
- nameEng :Text
- date :JFXDatePicker
- vbox :VBox
- scroll :ScrollPane
- cc :CinemaController
- uc :UserController
- rootPane :StackPane
- textNotHaveShowtime :Text
- hbox_showtimepertheatre :HBoox
+ loadShowtime(LocalDate) :void
+ getDate() :void
+ back(ActionEvent) :void
RegisterController
- rootPane :AnchorPane
- txtUsername :JFXTextField
- txtPassword :JFXTextField
- txtConfirmPassword :JFXTextField
- txtFirstName :JFXTextField
- txtLastname :JFXTextField
- txtEmail :JFXTextField
- labwlUsername :Label
- labelPassword :Label
- labelConfirmPassword :Label
- labelFirstname :Label
- labelLastname :Label
- labelEmail :Label
- allUser :List<User>
- uc :UserController
+ handleCreate(ActionEvent) :void
+ handleClear(ActionEvent) :void
+ clearForm() :void
+ clearLabel() :void
+ resetAllSyle() :void
+ handleBack(ActionEvent) :void
layoutMediumController
- F1 :ImageView
- F2 :ImageView
- F3 :ImageView
- F4 :ImageView
- F5 :ImageView
- F6 :ImageView
- F7 :ImageView
- F8 :ImageView
- F9 :ImageView
- F10 :ImageView
- F11 :ImageView
- F12 :ImageView
- F13 :ImageView
- F14 :ImageView
- F15 :ImageView
- F17 :ImageView
- F18 :ImageView
- F19 :ImageView
- F20 :ImageView
- E1 :ImageView
- E2 :ImageView
- E3 :ImageView
- E4 :ImageView
- E5 :ImageView
- E6 :ImageView
- E7 :ImageView
- E8 :ImageView
- E9 :ImageView
- E10 :ImageView
- E11 :ImageView
- E12 :ImageView
- E13 :ImageView
- E14 :ImageView
- E15 :ImageView
- E16 :ImageView
- E17 :ImageView
- E18 :ImageView
- E19 :ImageView
- E20 :ImageView
- D1 :ImageView
- D2 :ImageView
- D3 :ImageView
- D4 :ImageView
- D5 :ImageView
- D6 :ImageView
- D7 :ImageView
- D8 :ImageView
- D9 :ImageView
- D10 :ImageView
- D11 :ImageView
- D12 :ImageView
- D13 :ImageView
- D14 :ImageView
- D15 :ImageView
- D16 :ImageView
- D17 :ImageView
- D18 :ImageView
- D19 :ImageView
- D20 :ImageView
- C1 :ImageView
- C2 :ImageView
- C3 :ImageView
- C4 :ImageView
- C5 :ImageView
- C6 :ImageView
- C7 :ImageView
- C8 :ImageView
- C9 :ImageView
- C10 :ImageView
- C11 :ImageView
- C12 :ImageView
- C13 :ImageView
- C14 :ImageView
- C15 :ImageView
- C16 :ImageView
- C17 :ImageView
- C18 :ImageView
- C19 :ImageView
- C20 :ImageView
- B1 :ImageView
- B2 :ImageView
- B3 :ImageView
- B4 :ImageView
- B5 :ImageView
- B6 :ImageView
- B7 :ImageView
- B8 :ImageView
- B9 :ImageView
- B10 :ImageView
- B11 :ImageView
- B12 :ImageView

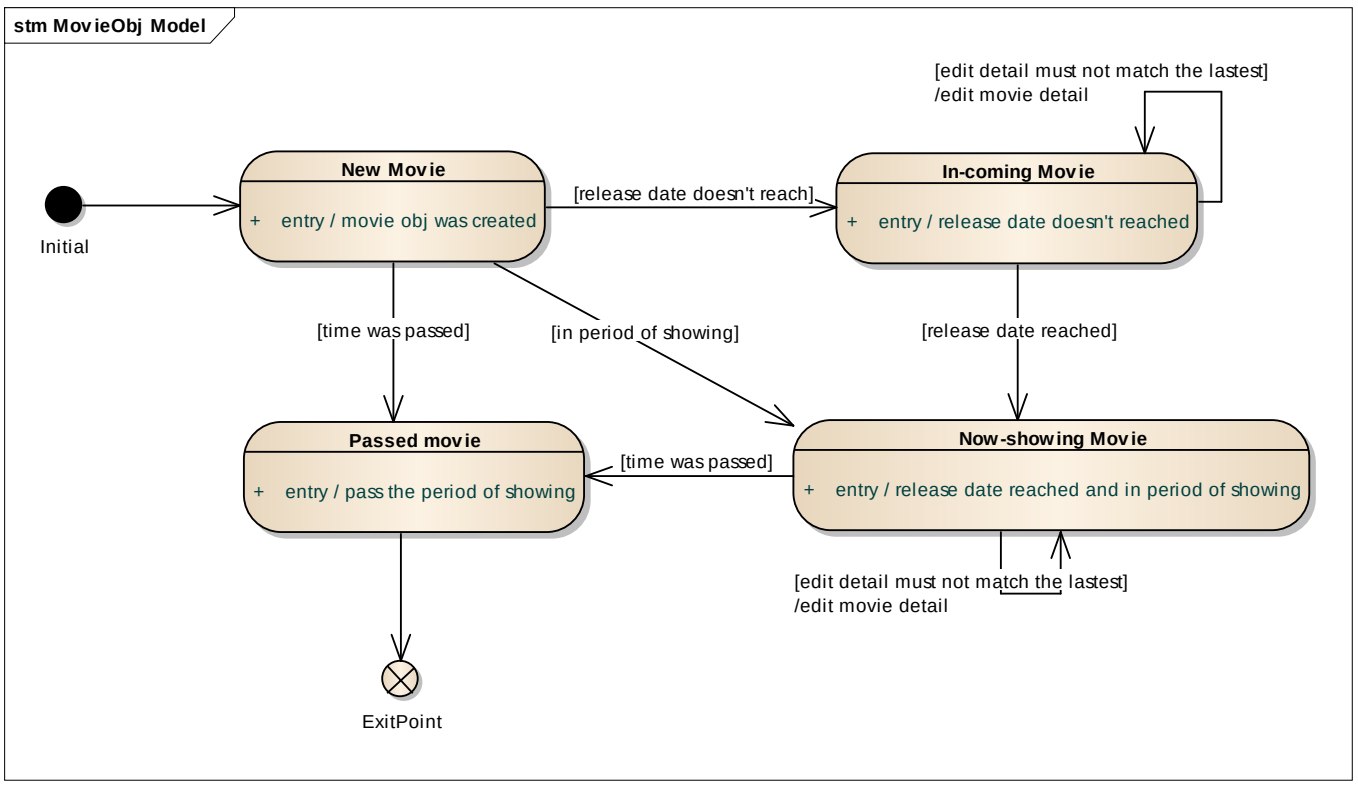

layoutMediumController
- F1 :ImageView
- F2 :ImageView
- F3 :ImageView
- F4 :ImageView
- F5 :ImageView
- F6 :ImageView
- F7 :ImageView
- F8 :ImageView
- F9 :ImageView
- F10 :ImageView
- F11 :ImageView
- F12 :ImageView
- F13 :ImageView
- F14 :ImageView
- F15 :ImageView
- F17 :ImageView
- F18 :ImageView
- F19 :ImageView
- F20 :ImageView
- E1 :ImageView
- E2 :ImageView
- E3 :ImageView
- E4 :ImageView
- E5 :ImageView
- E6 :ImageView
- E7 :ImageView
- E8 :ImageView
- E9 :ImageView
- E10 :ImageView
- E11 :ImageView
- E12 :ImageView
- E13 :ImageView
- E14 :ImageView
- E15 :ImageView
- E16 :ImageView
- E17 :ImageView
- E18 :ImageView
- E19 :ImageView
- E20 :ImageView
- D1 :ImageView
- D2 :ImageView
- D3 :ImageView
- D4 :ImageView
- D5 :ImageView
- D6 :ImageView
- D7 :ImageView
- D8 :ImageView
- D9 :ImageView
- D10 :ImageView
- D11 :ImageView
- D12 :ImageView
- D13 :ImageView
- D14 :ImageView
- D15 :ImageView
- D16 :ImageView
- D17 :ImageView
- D18 :ImageView
- D19 :ImageView
- D20 :ImageView
- C1 :ImageView
- C2 :ImageView
- C3 :ImageView
- C4 :ImageView
- C5 :ImageView
- C6 :ImageView
- C7 :ImageView
- C8 :ImageView
- C9 :ImageView
- C10 :ImageView
- C11 :ImageView
- C12 :ImageView
- C13 :ImageView
- C14 :ImageView
- C15 :ImageView
- C16 :ImageView
- C17 :ImageView
- C18 :ImageView
- C19 :ImageView
- C20 :ImageView
- B1 :ImageView
- B2 :ImageView
- B3 :ImageView
- B4 :ImageView
- B5 :ImageView
- B6 :ImageView
- B7 :ImageView
- B8 :ImageView
- B9 :ImageView
- B10 :ImageView
- B11 :ImageView
- B12 :ImageView
- B13 :ImageView
- B14 :ImageView
- B15 :ImageView
- B16 :ImageView
- B17 :ImageView
- B18 :ImageView
- B19 :ImageView
- B20 :ImageView
- A1 :ImageView
- A2 :ImageView
- A3 :ImageView
- A4 :ImageView
- A5 :ImageView
- rootPane :AncorPane
- count :int
- isFound :boolean
- maxIV :String
- minIV :String
- ivList :List<String>
- ivListAll :List<String<>
- ivListDisable :List<String>
- cc :CinemaControleer
- uc :UserController
- txtShowtime :Text
- labelPreSummary :Text
- txtMovie :Text
- btnNext :Button
+ initialize(URL, ResourceBundle) :void
+ seatAssign() :void
+ seatDisable() :void
+ nextPage(ActionEvent) :void
+ hndleButtonAction(ActionEvent) :void
+ back(ActionEvent) :void

Statechart diagram

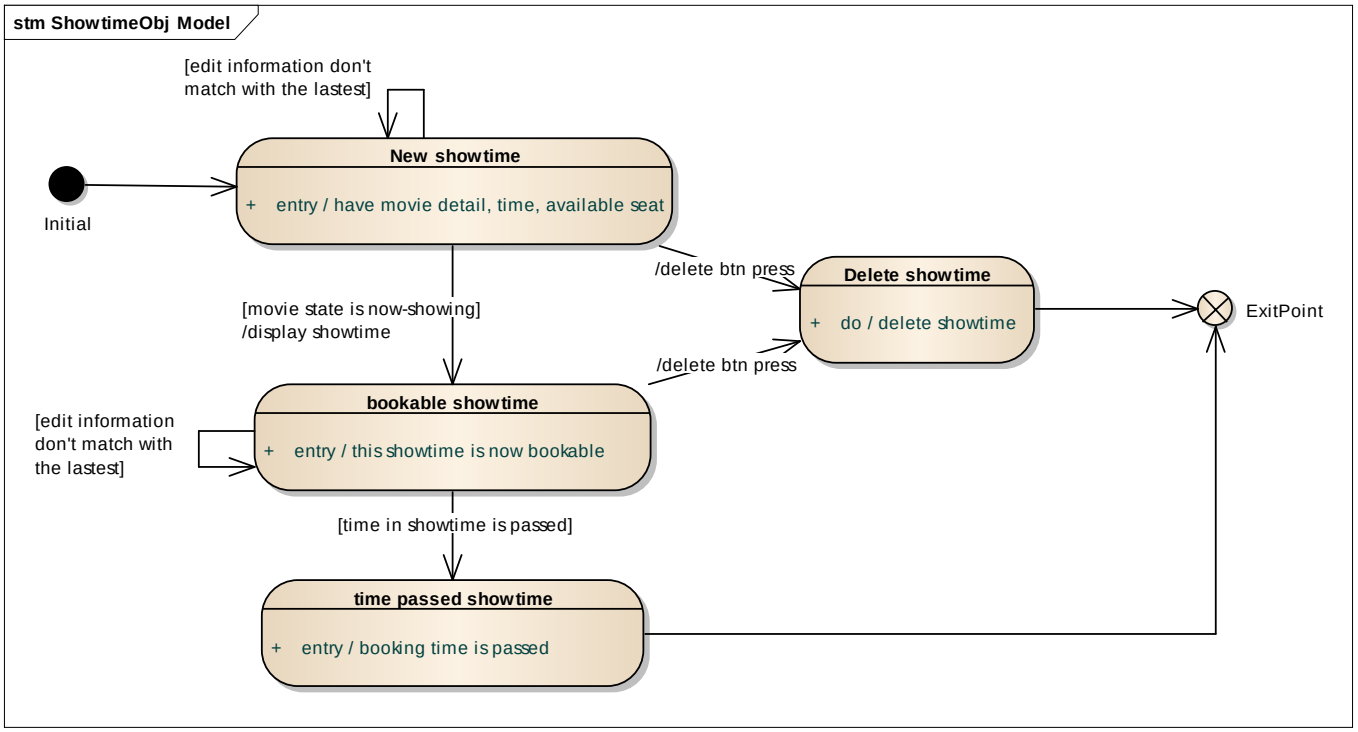
สำหรับการจอง และยกเลิก รอบฉายภาพยนตร์



สำหรับการเพิ่มภาพยนตร์



สำหรับการเพิ่ม แก้ไข และลบ รอบฉายภาพยนตร์



Chapter 4

Implementation

Source code

Class Booking

```
1.  package cinema;
2.
3.
4.      import cinema.ui.AlertMaker;
5.      import java.io.Serializable;
6.      import java.text.DateFormat;
7.      import java.text.SimpleDateFormat;
8.      import java.util.ArrayList;
9.      import java.util.Calendar;
10.     import java.util.List;
11.     import javax.persistence.Entity;
12.     import javax.persistence.FetchType;
13.     import javax.persistence.GeneratedValue;
14.     import javax.persistence.GenerationType;
15.     import javax.persistence.Id;
16.     import javax.persistence.OneToMany;
17.     import javax.persistence.OneToOne;
18.
19.
20.     /**
21.      *
22.      * @author pisit
23.      */
24.     @Entity
25.     public class Booking implements Serializable{
26.         private static final long serialVersionUID = 1L;
27.
28.         @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
29.         private int id;
30.
31.         @OneToOne(fetch= FetchType.EAGER)
32.         private Showtime showtime;
33.
34.         @OneToOne(fetch= FetchType.EAGER)
35.         private User user;
36.
37.         @OneToOne(fetch= FetchType.EAGER)
38.         private Promotion promotion = null;
39.
40.         private boolean isCancel = false;
41.
42.
43.     @OneToMany(fetch= FetchType.EAGER)
44.         private List<Seat> bookedSeatList = new ArrayList<Seat>();
45.
46.         private double totalCost;
47.
48.         String bookingCreateDatetime;
49.         String bookingUpdateDatetime;
50.
51.
52.         // No Promotion
53.     //    public Booking(Showtime showtime, List<Seat> seat, User user,double total
    Cost) {
54.     //        this.showtime = showtime;
55.     //        this.bookedSeatList = seat;
```

```
56.     //         this.user = user;
57.     //         this.totalCost = totalCost;
58.     //         this.isCancel = false;
59.     //         this.bookingCreateDatetime = getCurrentDatetime();
60.     //         this.bookingUpdateDatetime = getCurrentDatetime();
61.     //     }
62.
63.         public Booking(Showtime showtime, List<Seat> seat, User user, Promotion pro
    motion,double totalCost) {
64.             this.showtime = showtime;
65.             this.bookedSeatList = seat;
66.             this.user = user;
67.             this.promotion = promotion;
68.             this.totalCost = totalCost;
69.             this.isCancel = false;
70.             this.bookingCreateDatetime = getCurrentDatetime();
71.             this.bookingUpdateDatetime = getCurrentDatetime();
72.         }
73.
74.         public String getCurrentDatetime(){
75.             DateFormat dateFormat = new SimpleDateFormat("d MMMM yyyy HH:mm:ss");
76.             Calendar cal = Calendar.getInstance();
77.             return dateFormat.format(cal.getTime());
78.         }
79.         public void updateDatetime(){
80.             this.bookingUpdateDatetime = getCurrentDatetime();
81.         }
82.
83.
84.         public String getBookingCreateDatetime() {
85.             return bookingCreateDatetime;
86.         }
87.         public String getBookingUpdateDatetime() {
88.             return bookingUpdateDatetime;
89.
90.     }
91.         public String getShowtimeDetail(){
92.             return "("+ showtime.getId() + ") Name(EN) : " + showtime.getMovieEng()
     + "\n" +
93.                         "Name(TH) : " + showtime.getMovieThai()+ "\n" +
94.                         showtime.getSoundtrack()+"/"+showtime.getSubtitle()+ " (" +
    showtime.getSystem() + ")\n" +
95.                         "Time : " + showtime.getShowtime()+ "\n" +
96.                         "Theatre : " + showtime.getTheatre().getTheatreNumber();

97.         }
98.         public String getUserDetail(){
99.             return "(" + user.getId() + ") Name : " + user.getFirstname()+" "+ user
    .getLastname() + "\n" +
100.                            "Email : " + user.getEmail()+ "\n" +
101.                           "Money : " + user.getMoney();
102.                }
103.
104.                public String getBookedSeatString(){
105.                    String seatList = "";
106.                    for (Seat seat : bookedSeatList) {
107.                        seatList = seatList + "(" + seat.getId() + ") " + seat.getSe
    atName() + " : " + seat.getSeatStatus() +" " + seat.getSeatPrice()+ "\n";
108.                    }
109.                    return seatList;
110.                }
111.
112.
113.                public boolean getIsCancel() {
114.                    return isCancel;
115.                }
```

```
116.
117.                public String getStatus() {
118.                    if(!isCancel) {
119.                        return "Active";
120.                    }
121.                    else {
122.                        return "Cancelled";
123.                    }
124.                }
125.
126.                public boolean isCancel() {
127.                    return isCancel;
128.                }
129.                public void setIsCancel(boolean isCancel) {
130.                    this.isCancel = isCancel;
131.                }
132.
133.            public int getId() {
134.                    return id;
135.                }
136.                public Showtime getShowtime() {
137.                    return showtime;
138.                }
139.                public void setShowtime(Showtime showtime) {
140.                    this.showtime = showtime;
141.                }
142.                public User getUser() {
143.                    return user;
144.                }
145.                public void setUser(User user) {
146.                    this.user = user;
147.                }
148.                public Promotion getPromotion() {
149.                    return promotion;
150.                }
151.                public void setPromotion(Promotion promotion) {
152.                    this.promotion = promotion;
153.                }
154.                public List<Seat> getBookedSeatList() {
155.                    return bookedSeatList;
156.                }
157.                public void setBookedSeatList(List<Seat> bookedSeatList) {
158.                    this.bookedSeatList = bookedSeatList;
159.                }
160.                public double getTotalCost() {
161.                    return totalCost;
162.                }
163.                public void setTotalCost(double totalCost) {
164.                    this.totalCost = totalCost;
165.                }
166.                public double getTotalSeatPrice(){
167.                    double total = 0;
168.                    for (Seat seat : bookedSeatList) {
169.                        total += seat.getSeatPrice();
170.                    }
171.                    return total;
172.                }
173.
174.
175.
176.                @Override
177.                public String toString() {
178.                    return "Booking{\n" + "id=" + id +
179.                            "Showtime=" + showtime + "\n" +
180.                            "User=" + user + "\n" +
181.                            "Promotion=" + promotion + "\n" +
```

```
182.                         "BookedSeatList=" + bookedSeatList + "\n" +
183.                         "TotalCost=" + totalCost + '}';
184.             }
185.
186.             public double cancelBooking(){
187.                 double userReturn = this.totalCost - (this.totalCost * 0.1); //
    return to user 90%
188.                 this.totalCost = this.totalCost - userReturn; // booking will le
    ft only 10%
189.                 return userReturn;
190.             }
191.
192.
193.
194.
195.         }
```

Class Promotion

```
1.  package cinema;
2.
3.
4.      import java.io.Serializable;
5.      import javax.persistence.Entity;
6.      import javax.persistence.GeneratedValue;
7.      import javax.persistence.GenerationType;
8.      import javax.persistence.Id;
9.
10.
11.     /**
12.      *
13.      * @author pisit
14.      */
15.
16.
17.     @Entity
18.     public class Promotion implements Serializable{
19.         private static final long serialVersionUID = 1L;
20.
21.
22.         @Id @GeneratedValue
23.         private int promotionID;
24.
25.         private String name;
26.         private String description;
27.         private String remainingDate;
28.         private double discount = 0;
29.
30.
31.         public Promotion(String name, String description, String remainingDate, Dou
    ble discount) {
32.             this.name = name;
33.             this.description = description;
34.             this.remainingDate = remainingDate;
35.             this.discount = discount;
36.         }
37.
38.
39.         public int getPromotionID() {
40.             return promotionID;
41.         }
42.         public void setPromotionID(int promotionID) {
43.             this.promotionID = promotionID;
44.         }
45.         public String getName() {
46.             return name;
47.         }
48.         public void setName(String name) {
49.             this.name = name;
50.         }
51.         public String getDescription() {
52.             return description;
53.         }
54.         public void setDescription(String description) {
55.             this.description = description;
56.         }
57.         public String getRemainingDate() {
58.             return remainingDate;
59.         }
60.         public void setRemainingDate(String remainingDate) {
61.             this.remainingDate = remainingDate;
```

```
62.         }
63.         public double getDiscount() {
64.             return discount;
65.         }
66.         public void setDiscount(double discount) {
67.             this.discount = discount;
68.         }
69.
70.
71.         @Override
72.         public String toString() {
73.             return ("Promotion ID : "+ this.getPromotionID()+
74.                     "\nName : " + this.getName()+
75.                     "\nDescription : " + this.getDescription()+
76.                     "\nRemainingDate : " + this.getRemainingDate()+
77.                     "\nDiscount : " + this.getDiscount() + "\n"
78.                     );
79.         }
80.
81.
82.
83.
84.
85.     }
```

Class PromotionController

```
1.  package cinema;
2.
3.  import java.time.LocalDate;
4.  import java.time.format.DateTimeFormatter;
5.  import java.time.temporal.ChronoUnit;
6.  import java.util.ArrayList;
7.  import java.util.List;
8.  import javax.persistence.EntityManager;
9.  import javax.persistence.EntityManagerFactory;
10. import javax.persistence.Persistence;
11. import javax.persistence.Query;
12.
13. public class PromotionController {
14.     private PromotionController() {}
15.     private static PromotionController instance = new PromotionController();
16.     public static PromotionController getInstance() {
17.         return instance;
18.     }
19.
20.     public EntityManagerFactory emf;
21.     public EntityManager em;
22.     public void openConnection(){
23.         emf = Persistence.createEntityManagerFactory("db/CinemaODB.odb"); // connec
    t to object database file
24.     em = emf.createEntityManager();
25.     }
26.     public void closeConnection(){
27.         em.close();
28.         emf.close();
29.     }
30.
31.     // Operation
32.     public Promotion getPromotion(int id){
33.         openConnection();
34.         // find object
35.         em.getTransaction().begin(); // start connection
36.         Promotion promotion = em.find(Promotion.class, id);
37.         // Close the database connection:
38.         em.getTransaction().commit(); // add all persist to database
39.         closeConnection();
40.         return promotion;
41.     }
42.     public void addPromotion(Promotion promotion){
43.         openConnection();
44.     em.getTransaction().begin(); // start connection
45.         em.persist(promotion); // add user to persist
46.         em.getTransaction().commit(); // add all persist to database
47.         closeConnection();
48.     }
49.     public void editPromotion(int id, Promotion editPromotion){
50.         openConnection();
51.         // find object
52.         Promotion promotion = em.find(Promotion.class, id);
53.         em.getTransaction().begin();
54.         promotion.setName(editPromotion.getName());
55.         promotion.setDescription(editPromotion.getDescription());
56.         promotion.setDiscount(editPromotion.getDiscount());
57.         promotion.setRemainingDate(editPromotion.getRemainingDate());
58.         em.getTransaction().commit();
59.         closeConnection();
60.     }
61.     public void deletePromotion(int id){
```

```
62.         openConnection();
63.         em.getTransaction().begin(); // start connection
64.         Query query = em.createQuery("DELETE FROM Promotion p WHERE p.promotionID =
    :id");
65.         query.setParameter("id", id).executeUpdate();
66.         em.getTransaction().commit(); // add all persist to database
67.         closeConnection();
68.     }
69.
70.     // Get all
71.     private List<Promotion> promotionList = new ArrayList<Promotion>();
72.     public void updatePromotionList() {
73.         openConnection();
74.         em.getTransaction().begin(); // start connection
75.         Query query = em.createQuery("SELECT p FROM Promotion p");
76.         List<Promotion> promotion = query.getResultList(); // get movie
77.         this.promotionList = promotion;
78.         closeConnection();
79.     }
80.     public List<Promotion> getPromotionList() {
81.         this.updatePromotionList();
82.         return promotionList;
83.     }
84.
85.     // Search
86.     private List<Promotion> promotionSearchList = new ArrayList<Promotion>();
87.     public void searchPromotion(String text, String searchOf){
88.         // ดึงข้อมูลจาก db ส่วน User
89.         openConnection();
90.         em.getTransaction().begin(); // start connection
91.         Query querySearch = em.createQuery("SELECT p FROM Promotion p WHERE lower(p
    ." + searchOf +") LIKE lower(:" + searchOf +")", Promotion.class);
92.         querySearch.setParameter(searchOf.toString(), "%"+text+"%");
93.         List<Promotion> promotions = querySearch.getResultList(); // get user
94.         this.promotionSearchList = promotions;
95.         closeConnection();
96.     }
97.     public List<Promotion> getPromotionSearchList(){
98.         return this.promotionSearchList;
99.     }
100.
101.        public List<Promotion> getActivePromotionList() {
102.            this.updatePromotionList();
103.
104.            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMMM yy
    yy");
105.
106.            List<Promotion> activePromotion = new ArrayList<Promotion>();
107.            for (Promotion promotion : getPromotionList()) {
108.                if((LocalDate.now().until(LocalDate.parse(promotion.getRemaining
    Date(),formatter), ChronoUnit.DAYS) >= 0)){
109.                    activePromotion.add(promotion);
110.                }
111.            }
112.            return activePromotion;
113.        }
114.
115.        public List<Promotion> getUnusePromotion(int userId){
116.            List<Promotion> unuseList = new ArrayList<Promotion>();
117.            openConnection();
118.            // find object
119.            em.getTransaction().begin(); // start connection
120.            User user = em.find(User.class, userId);
121.
122.            List<Promotion> userP = user.getPromotionList();
123.            List<Integer> userPInt = new ArrayList<Integer>();
```

```
124.                for (Promotion promotion : userP) {
125.                    userPInt.add(promotion.getPromotionID());
126.                }
127.
128.                List<Promotion> allP = getActivePromotionList();
129.                List<Integer> allPInt = new ArrayList<Integer>();
130.                for (Promotion promotion : allP) {
131.                    allPInt.add(promotion.getPromotionID());
132.                }
133.
134.                for (Integer integer : userPInt) {
135.                    allPInt.remove(integer);
136.                }
137.
138.                for (Integer integer : allPInt) {
139.                    unuseList.add(getPromotion(integer));
140.                }
141.                return unuseList;
142.
143.            }
144.
145.        }
```

Class Movie

```
1.  package cinema;
2.
3.  import java.io.Serializable;
4.  import javax.persistence.Entity;
5.  import javax.persistence.GeneratedValue;
6.  import javax.persistence.GenerationType;
7.  import javax.persistence.Id;
8.
9.  @Entity
10. public class Movie implements Serializable{
11.     private static final long serialVersionUID = 1L;
12.
13.     @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
14.     private int id;
15.
16.     private String englishName;
17.     private String thaiName;
18.     private String director;
19.     private String cast;
20.     private String synopsis;
21.     private String genre;
22.     private String releaseDate;
23.     private String time;
24.     private String poster;
25.     private String trailer;
26.
27.     public Movie(String englishName, String thaiName, String director, String cast,
    String synopsis, String genre, String time, String releaseDate, String poster, Str
    ing trailer) {
28.         this.englishName = englishName;
29.         this.thaiName = thaiName;
30.         this.director = director;
31.         this.cast = cast;
32.         this.synopsis = synopsis;
33.         this.genre = genre;
34.         this.time = time;
35.         this.releaseDate = releaseDate;
36.         this.poster = poster;
37.         this.trailer = trailer;
38.     }
39.
40.     public int getId() {
41.         return id;
42.     }
43.     public void setId(int id) {
44.         this.id = id;
45.     }
46.     public String getEnglishName() {
47.         return englishName;
48.     }
49.     public void setEnglishName(String englishName) {
50.         this.englishName = englishName;
51.     }
52.     public String getThaiName() {
53.         return thaiName;
54.     }
55.     public void setThaiName(String thaiName) {
56.         this.thaiName = thaiName;
57.     }
58.     public String getDirector() {
59.         return director;
60.     }
```

```
61.     public void setDirector(String director) {
62.         this.director = director;
63.     }
64.     public String getCast() {
65.         return cast;
66.     }
67.     public void setCast(String cast) {
68.         this.cast = cast;
69.     }
70.     public String getSynopsis() {
71.         return synopsis;
72.     }
73.     public void setSynopsis(String synopsis) {
74.         this.synopsis = synopsis;
75.     }
76.     public String getGenre() {
77.         return genre;
78.     }
79.     public void setGenre(String genre) {
80.         this.genre = genre;
81.     }
82.     public String getReleaseDate() {
83.         return releaseDate;
84.     }
85.     public void setReleaseDate(String releaseDate) {
86.         this.releaseDate = releaseDate;
87.     }
88.     public String getTime() {
89.         return time;
90.     }
91.
92.     public String getTimeMinute() {
93.         String[] timeMinute = new String[2];
94.         timeMinute[0] = time.substring(0, 2);
95.         timeMinute[1] = time.substring(3);
96.         int times = (Integer.parseInt(timeMinute[0]) * 60) + Integer.parseInt(timeM
    inute[1]);
97.         String returnTime = times + "";
98.         return returnTime;
99.     }
100.            public void setTime(String time) {
101.                this.time = time;
102.            }
103.            public String getPoster() {
104.                return poster;
105.            }
106.            public void setPoster(String poster) {
107.                this.poster = poster;
108.            }
109.            public String getTrailer() {
110.                return trailer;
111.            }
112.            public void setTrailer(String trailer) {
113.                this.trailer = trailer;
114.            }
115.
116.            //-----------------------------------------------------------
    - Function Method
117.
118.            @Override
119.            public String toString()
120.            {
121.                return ("Movie ID : "+ this.getId()+
122.                        "\nEnglish Name : " + this.getEnglishName() +
123.                        "\nThai Name : "+ this.getThaiName() +
124.                        "\nDirector : " + this.getDirector() +
```

```
125.                    "\nCast : " + this.getCast() +
126.                    "\nSynopsis : " + this.getSynopsis() +
127.                    "\nGenre : " + this.getGenre() +
128.                    "\nTime : " + this.getTime() +
129.                    "\nRelease Date : " + this.getReleaseDate() +
130.                    "\nPoster : " + this.getPoster() +
131.                    "\nTrailer : " + this.getTrailer());
132.            }
133.
134.
135.        }
```

Class Seat

```
1.  package cinema;
2.
3.
4.      import java.io.Serializable;
5.      import javax.persistence.Entity;
6.      import javax.persistence.GeneratedValue;
7.      import javax.persistence.GenerationType;
8.      import javax.persistence.Id;
9.      import javax.persistence.ManyToOne;
10.
11.
12.     @Entity
13.     public abstract class Seat implements Serializable{
14.         private static final long serialVersionUID = 1L;
15.
16.         @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
17.         protected int id;
18.
19.         @ManyToOne
20.         protected Showtime showtime; // เป็นของ showtime อะไร
21.
22.         protected String seatName;
23.         protected String seatType;
24.         protected double seatPrice;
25.         protected boolean seatStatus;
26.
27.         protected double basePrice = 120;
28.
29.         abstract public double getSeatPrice();
30.         abstract void setSeatPrice(double seatPrice);
31.         abstract public String getSeatType();
32.         // set for which showtime?
33.         abstract public void setShowtime(Showtime showtime);
34.         abstract public Showtime getShowtime();
35.         // set and get status of seat
36.         abstract void setSeatStatus(boolean status);
37.         abstract public boolean getSeatStatus();
38.         // get seat id
39.         abstract public int getId();
40.         // get seat name
41.         abstract public String getSeatName();
42.     }
```

Class NormalSeat

```
1.  package cinema;
2.
3.
4.      import javax.persistence.Entity;
5.
6.
7.      /**
8.       *
9.       * @author pisit
10.      */
11.     @Entity
12.     public class NormalSeat extends Seat {
13.
14.         public NormalSeat(String name, Showtime showtime, double increasePrice) {
15.             this.seatName = name;
16.             this.seatType = "Normal";
17.             this.seatPrice = this.basePrice + increasePrice;
18.             this.showtime = showtime;
19.         }
20.
21.
22.         @Override
23.         public double getSeatPrice() {
24.             return this.seatPrice;
25.         }
26.
27.
28.         @Override
29.         public void setSeatPrice(double increasePrice) {
30.             this.seatPrice = this.basePrice + increasePrice;
31.         }
32.
33.
34.         @Override
35.         public String getSeatType() {
36.             return this.seatType;
37.         }
38.
39.
40.         @Override
41.         public void setShowtime(Showtime showtime) {
42.             this.showtime = showtime;
43.         }
44.
45.
46.         @Override
47.         public Showtime getShowtime() {
48.             return this.showtime;
49.         }
50.
51.
52.         @Override
53.         public void setSeatStatus(boolean status) {
54.             this.seatStatus = status;
55.         }
56.
57.
58.         @Override
59.         public boolean getSeatStatus() {
60.             return this.seatStatus;
61.         }
62.
```

```
63.         @Override
64.         public String toString() {
65.             return this.id + " " + this.seatName + " : (" + this.seatStatus + ") :
    " + this.seatType + ", Price = " + this.seatPrice + "\n";
66.         }
67.
68.         @Override
69.         public String getSeatName() {
70.             return this.seatName;
71.         }
72.
73.
74.         @Override
75.         public int getId() {
76.             return this.id;
77.         }
78.     }
```

Class HoneymoonSeat

```
1.  package cinema;
2.
3.
4.      import javax.persistence.Entity;
5.
6.
7.      /**
8.       *
9.       * @author pisit
10.      */
11.     @Entity
12.     public class HoneymoonSeat extends Seat {
13.
14.
15.
16.         public HoneymoonSeat(String name, Showtime showtime, double increasePrice)
    {
17.             this.seatName = name;
18.             this.seatType = "Honeymoon";
19.             this.seatPrice = this.basePrice + 30 + increasePrice;
20.             this.showtime = showtime;
21.         }
22.
23.
24.         @Override
25.         public double getSeatPrice() {
26.             return this.seatPrice;
27.         }
28.
29.
30.         @Override
31.         public void setSeatPrice(double increasePrice) {
32.             this.seatPrice = this.basePrice + 30 + increasePrice;
33.         }
34.
35.
36.         @Override
37.         public String getSeatType() {
38.             return this.seatType;
39.         }
40.
41.
42.         @Override
43.         public void setShowtime(Showtime showtime) {
44.             this.showtime = showtime;
45.         }
46.
47.
48.         @Override
49.         public Showtime getShowtime() {
50.             return this.showtime;
51.         }
52.
53.
54.         @Override
55.         public void setSeatStatus(boolean status) {
56.             this.seatStatus = status;
57.         }
58.
59.
60.         @Override
61.         public boolean getSeatStatus() {
```

```
62.             return this.seatStatus;
63.         }
64.
65.         @Override
66.         public String toString() {
67.             return this.id + " " + this.seatName + " : (" + this.seatStatus + ") :
    " + this.seatType + ", Price = " + this.seatPrice + "\n";
68.         }
69.
70.         @Override
71.         public String getSeatName() {
72.             return this.seatName;
73.         }
74.
75.
76.         @Override
77.         public int getId() {
78.             return this.id;
79.         }
80.     }
```

Class PairedSeat

```
1.  package cinema;
2.
3.
4.      import javax.persistence.Entity;
5.
6.
7.      /**
8.       *
9.       * @author pisit
10.      */
11.     @Entity
12.     public class PairedSeat extends Seat {
13.
14.         public PairedSeat(String name, Showtime showtime, double increasePrice) {
15.             this.seatName = name;
16.             this.seatType = "Paired";
17.             this.seatPrice = this.basePrice + 100 + increasePrice;
18.             this.showtime = showtime;
19.         }
20.
21.
22.
23.         @Override
24.         public double getSeatPrice() {
25.             return this.seatPrice;
26.         }
27.
28.
29.         @Override
30.         public void setSeatPrice(double increasePrice) {
31.             this.seatPrice = this.basePrice + 100 + increasePrice;
32.         }
33.
34.
35.         @Override
36.         public String getSeatType() {
37.             return this.seatType;
38.         }
39.
40.
41.         @Override
42.         public void setShowtime(Showtime showtime) {
43.             this.showtime = showtime;
44.         }
45.
46.
47.         @Override
48.         public Showtime getShowtime() {
49.             return this.showtime;
50.         }
51.
52.
53.         @Override
54.         public void setSeatStatus(boolean status) {
55.             this.seatStatus = status;
56.         }
57.
58.
59.         @Override
60.         public boolean getSeatStatus() {
61.             return this.seatStatus;
62.         }
```

```
63.
64.         @Override
65.         public String toString() {
66.             return this.id + " " + this.seatName + " : (" + this.seatStatus + ") :
    " + this.seatType + ", Price = " + this.seatPrice + "\n";
67.         }
68.
69.         @Override
70.         public String getSeatName() {
71.             return this.seatName;
72.         }
73.
74.
75.         @Override
76.         public int getId() {
77.             return this.id;
78.         }
79.
80.     }
```

Class Showtime

```
1.  package cinema;
2.
3.  import cinema.Movie;
4.  import cinema.Seat;
5.  import cinema.Theatre;
6.
7.  import java.io.Serializable;
8.  import java.text.ParseException;
9.  import java.text.SimpleDateFormat;
10. import java.util.ArrayList;
11. import java.util.Calendar;
12. import java.util.Date;
13. import java.util.List;
14. import javax.persistence.CascadeType;
15. import javax.persistence.Entity;
16. import javax.persistence.FetchType;
17. import javax.persistence.GeneratedValue;
18. import javax.persistence.GenerationType;
19. import javax.persistence.Id;
20. import javax.persistence.OneToOne;
21.
22. /**
23.  *
24.  * @author pisit
25.  */
26.
27. @Entity
28. public class Showtime implements Serializable{
29.     private static final long serialVersionUID = 1L;
30.
31.     @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
32.     private int id;
33.
34.     @OneToOne(fetch= FetchType.EAGER)
35.     private Movie movie; // เก็บหนังเรื่องอะไร
36.     @OneToOne(fetch= FetchType.EAGER)
37.     private Theatre theatre; // โรงไหน
38.
39.     private double increaseSeatPrice = 0;
40.     private String system = "2D";
41.     private String soundtrack = "TH";
42.     private String subtitle = "-";
43.     private String startTime = "00:00";
44.     private String date;
45.
46.     //@OneToOne(cascade=CascadeType.PERSIST)
47.     @OneToOne(mappedBy = "showtime",cascade = CascadeType.ALL,orphanRemoval = true)

48.     private List<Seat> seatList = new ArrayList<Seat>();
49.
50.     //private List<Seat> bookedList = new ArrayList<Seat>();
51.
52.     public Showtime(Movie movie, Theatre theatre, String system, String soundtrack,
     String date, String startTime, double increaseSeatPrice) {
53.         this.movie = movie;
54.         this.theatre = theatre;
55.         this.system = system;
56.         this.soundtrack = soundtrack;
57.         this.subtitle = chooseSubtitle(soundtrack);
58.
59.         this.startTime = startTime;
60.         this.date = date;
```

```
61.         this.increaseSeatPrice = increaseSeatPrice;
62.
63.         createSeatLayout(increaseSeatPrice);
64.     }
65.
66.     public String getDate() {
67.         return date;
68.     }
69.     public void setDate(String date) {
70.         this.date = date;
71.     }
72.
73.     public String chooseSubtitle(String soundtrack){
74.         String sub = "TH";
75.         if (this.soundtrack.toUpperCase().equals("TH")) { // ถ้าเป็น Thai ไม่มีซับ
76.             sub = "-";
77.         }
78.         return sub;
79.     }
80.
81.     // Seat operation one showtime to many seat
82.     public void addNormalSeat(int size, String name, double increaseSeatPrice){
83.         for(int i = 1; i <= size; i++) {
84.             seatList.add(new NormalSeat(name+i,this,increaseSeatPrice));
85.         }
86.     }
87.     public void addHoneymoonSeat(int size, String name, double increaseSeatPrice){

88.         for(int i = 1; i <= size; i++) {
89.             seatList.add(new HoneymoonSeat(name+i,this,increaseSeatPrice));
90.         }
91.     }
92.     public void addPairedSeat(int size, String name, double increaseSeatPrice){
93.         for(int i = 1; i <= size; i++) {
94.             seatList.add(new PairedSeat(name+i,this,increaseSeatPrice));
95.         }
96.     }
97.
98.     public final void createSeatLayout(double increaseSeatPrice) {
99.         /* Create Seat Layout */
100.             // Create Pair Seat for 1 row, 5 columns
101.             addPairedSeat(5,"A",increaseSeatPrice);
102.             // Create Honeymoon Seat for 2 rows, each row contains 20 seats
103.             addHoneymoonSeat(20,"B",increaseSeatPrice);
104.             addHoneymoonSeat(20,"C",increaseSeatPrice);
105.
106.             // Create Normal Seat for 3 rows, each row contains 20 seats
107.             addNormalSeat(20,"D",increaseSeatPrice);
108.             addNormalSeat(20,"E",increaseSeatPrice);
109.             addNormalSeat(20,"F",increaseSeatPrice);
110.
111.         }
112.         public List<Seat> getSeatList() {return seatList;}
113.
114.         public void setSeatPrice(double increaseSeatPrice){
115.             for (Seat seat : seatList) {
116.                 seat.setSeatPrice(increaseSeatPrice);
117.             }
118.         }
119.         public void setSeatStatus(String seatNumber,boolean seatStatus){
120.             for (Seat seat : seatList) {
121.                 if(seat.getSeatName().equals(seatNumber)){
122.                     seat.setSeatStatus(seatStatus);
123.                 }
124.             }
125.         }
```

```
126.            public Seat getSeat(String seatNumber){
127.                Seat s = null;
128.                for (Seat seat : seatList) {
129.                    if(seat.getSeatName().equals(seatNumber)){
130.                       s = seat;
131.                    }
132.                }
133.                return s;
134.            }
135.
136.            public String getShowtimeDetail(){
137.                return "("+ getId() + ") " + getMovieEng() + "  " +
138.                            getSoundtrack()+"/"+getSubtitle()+ " (" + getSystem() + "
    )   " +
139.
140.                            "Theatre : " + getTheatre().getTheatreNumber();
141.            }
142.
143.            //-------------------------------------------------------------------
    - Method
144.            //************************** setter & getter**************************
    ****************************
145.            public void setMovie(Movie movie) {this.movie = movie;}
146.            public Movie getMovie() {return movie;}
147.
148.            public void setTheatre(Theatre theatre) {this.theatre = theatre;}
149.            public Theatre getTheatre() {return theatre;}
150.
151.            public void setSystem(String system) {this.system = system;}
152.            public String getSystem() {return system;}
153.
154.            public void setSoundtrack(String soundtrack) {
155.                this.soundtrack = soundtrack;
156.                this.subtitle = chooseSubtitle(soundtrack);
157.            }
158.            public String getSoundtrack() {return soundtrack;}
159.
160.            public void setSubtitle(String subtitle) {this.subtitle = subtitle;}
161.            public String getSubtitle() {return subtitle;}
162.
163.            public void setStartTime(String startTime) {this.startTime = startTime;}

164.            public String getOnlyStartTime(){
165.                return this.startTime;
166.            }
167.
168.            public String getStartTime() {return getShowtime();}
169.
170.            public void setIncreaseSeatPrice(double increaseSeatPrice) {this.increas
    eSeatPrice = increaseSeatPrice;}
171.            public double getIncreaseSeatPrice(String id) {return increaseSeatPrice;
    }
172.            public double getIncreaseSeatPrice(int id) {return increaseSeatPrice;}
173.            public double getIncreaseSeatPrice() {return increaseSeatPrice;}
174.
175.            public int getTheatreNo() {return theatre.getTheatreNumber();}
176.
177.            public void setId(int id) {this.id = id;}
178.            public int getId() {return id;}
179.
180.            public String getMovieEng() {return movie.getEnglishName();}
181.            public String getMovieThai() {return movie.getThaiName();}
182.
183.
184.            public boolean checkValidShowtime(String check) {
185.                return check.equalsIgnoreCase(startTime);
```

```
186.            }
187.
188.            //public double getPrice(String id) {return increaseSeatPrice + theatre.
    getSeatPrice(id);}
189.
190.            public int getTimeMovie() {return Integer.parseInt(movie.getTime());}
191.
192.
193.            public String getShowtime() {
194.                String myTime = this.startTime;
195.
196.                SimpleDateFormat df = new SimpleDateFormat("HH:mm");
197.                Date d = null;
198.                try {
199.                    d = df.parse(myTime);
200.                } catch (ParseException ex) {
201.                    System.out.println(ex);
202.                }
203.                Calendar cal = Calendar.getInstance();
204.                cal.setTime(d);
205.                int hour = Integer.parseInt(movie.getTime().substring(0, 2)) * 60;
206.                int minute = Integer.parseInt(movie.getTime().substring(3));
207.                int periodTime = hour + minute;
208.        //          System.out.println(hour);
209.        //          System.out.println(minute);
210.        //          System.out.println(periodTime);
211.
212.                cal.add(Calendar.MINUTE, periodTime);
213.                String newTime = df.format(cal.getTime());
214.                return (myTime + " - " +newTime);
215.            }
216.
217.        //    @Override
218.        //    public String toString() {
219.        //        return "Showtime{" + "\n" +
220.        //                "movie=" + movie + "\n ================= \n" +
221.        //              ", theatre=" + theatre + "\n ================= \n" +
222.        //              ", id=" + id + "\n" +
223.        //              ", increaseSeatPrice=" + increaseSeatPrice + "\n" +
224.        //              ", system=" + system + "\n" +
225.        //              ", soundtrack=" + soundtrack + "\n" +
226.        //              ", subtitle=" + subtitle + "\n" +
227.        //              ", startTime=" + startTime + "\n" ;
228.        //          //   ", seatList=" + seatList  ;
229.        //         //    + ", bookedList=" + bookedList + '}';
230.        //    }
231.
232.
233.
234.        }
```

Class Theatre

```
1.  package cinema;
2.
3.
4.      import java.io.Serializable;
5.      import java.util.ArrayList;
6.      import java.util.Comparator;
7.      import java.util.List;
8.      import java.util.concurrent.atomic.AtomicInteger;
9.      import javax.persistence.CascadeType;
10.     import javax.persistence.Entity;
11.     import javax.persistence.FetchType;
12.     import javax.persistence.GeneratedValue;
13.     import javax.persistence.GenerationType;
14.     import javax.persistence.Id;
15.     import javax.persistence.OneToMany;
16.     import javax.persistence.OneToOne;
17.     /**
18.      *
19.      * @author pisit
20.      */
21.     @Entity
22.     public class Theatre implements Serializable{
23.         private static final long serialVersionUID = 1L;
24.
25.         @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
26.         private int id;
27.
28.         private int theatreNumber;
29.         //private String theatreSize = "Medium"; // Default size is medium
30.         @OneToMany(fetch= FetchType.EAGER)
31.         private List<Showtime> showtimeList = new ArrayList<Showtime>(); // Display
    updated showtime when booking
32.
33.         private String theatreType;
34.
35.         public Theatre(int theatreNumber,String theatreType) {
36.             this.theatreNumber = theatreNumber;
37.             this.theatreType = theatreType;
38.         }
39.
40.
41.         public String getTheatreType() {
42.             return theatreType;
43.         }
44.         public void setTheatreType(String theatreType) {
45.             this.theatreType = theatreType;
46.         }
47.
48.         public void addShowtime(Showtime showtime){
49.             this.showtimeList.add(showtime);
50.         }
51.         public void deleteShowtime(Showtime st){
52.             showtimeList.remove(st);
53.         }
54.
55.         public List<Showtime> getShowtimeList(){
56.             this.showtimeList.sort(new Comparator<Showtime>() {
57.                 public int compare(Showtime o1, Showtime o2) {
58.                     // compare two instance of `Score` and return `int` as result.

59.                     return o1.getStartTime().compareTo(o2.getStartTime());
60.                 }
```

```
61.             });
62.             return this.showtimeList;
63.         }
64.
65.
66.         public void setShowtimeList(List<Showtime> showtimeList) {
67.             this.showtimeList = showtimeList;
68.         }
69.
70.         public int getId() {return id;}
71.         public void setId(int id) {this.id = id;}
72.
73.         public void setTheatreNumber(int theatreNumber) {this.theatreNumber = theat
    reNumber;}
74.         public int getTheatreNumber() {return theatreNumber;}
75.
76.
77.
78.
79.         @Override
80.         public String toString() {
81.             return "Theatre number : " + this.theatreNumber + "\nType : " + this.th
    eatreType;
82.         }
83.     }
```

Class User

```
1.  package cinema;
2.
3.
4.      import java.io.Serializable;
5.      import java.security.MessageDigest;
6.      import java.security.NoSuchAlgorithmException;
7.      import java.util.ArrayList;
8.      import java.util.List;
9.      import java.util.logging.Level;
10.     import java.util.logging.Logger;
11.     import javax.persistence.Entity;
12.     import javax.persistence.FetchType;
13.     import javax.persistence.GeneratedValue;
14.     import javax.persistence.Id;
15.     import javax.persistence.OneToMany;
16.
17.
18.     @Entity
19.     public class User implements Serializable{
20.         private static final long serialVersionUID = 1L;
21.
22.         @Id @GeneratedValue
23.         private int id;
24.
25.         private String username;
26.         private String password;
27.         private String firstname;
28.         private String lastname;
29.         private String email;
30.         private String type;
31.         private double money;
32.
33.         @OneToMany(fetch= FetchType.EAGER)
34.         private List<Promotion> promotionList = new ArrayList<Promotion>();
35.
36.         public User(String username, String password, String firstname, String last
    name, String email, String type, Double money) {
37.             this.username = username;
38.             this.password = password;
39.             this.firstname = firstname;
40.             this.lastname = lastname;
41.             this.email = email;
42.             this.type = type;
43.             this.money = money;
44.         }
45.
46.
47.         public void setEncryptPassword(){
48.             this.password = encryptPassword(this.password);
49.         }
50.
51.         public static String getEncryptPassword(String pass){
52.             return encryptPassword(pass);
53.         }
54.
55.         public static String encryptPassword(String password){
56.             String generatedPassword = null;
57.             try {
58.                 MessageDigest md = MessageDigest.getInstance("MD5");
59.                 // Add password bytes to digest
60.                 md.update(password.getBytes());
61.                 // Get the hash's bytes
```

```
62.                 byte[] bytes = md.digest();
63.                 // This bytes[] has bytes in decimal format;
64.                 // Convert it to hexadecimal format
65.                 StringBuilder sb = new StringBuilder();
66.                 for(int i=0; i< bytes.length ;i++)
67.                 {
68.                     sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).subst
    ring(1));
69.                 }
70.                 // Get complete hashed password in hex format
71.                 generatedPassword = sb.toString();
72.             } catch (NoSuchAlgorithmException ex) {
73.                 Logger.getLogger(User.class.getName()).log(Level.SEVERE, null, ex);

74.             }
75.             return generatedPassword;
76.         }
77.         public int getId() {
78.             return id;
79.         }
80.         public void setId(int id) {
81.             this.id = id;
82.         }
83.         public String getUsername() {
84.             return username;
85.         }
86.         public void setUsername(String username) {
87.             this.username = username;
88.         }
89.         public String getPassword() {
90.
91.         return password;
92.         }
93.         public void setPassword(String password) {
94.             this.password = password;
95.         }
96.         public String getFirstname() {
97.             return firstname;
98.         }
99.         public void setFirstname(String firstname) {
100.                this.firstname = firstname;
101.            }
102.            public String getLastname() {
103.                return lastname;
104.            }
105.            public void setLastname(String lastname) {
106.                this.lastname = lastname;
107.            }
108.            public String getEmail() {
109.                return email;
110.            }
111.            public void setEmail(String email) {
112.                this.email = email;
113.            }
114.            public String getType() {
115.                return type;
116.            }
117.            public void setType(String type) {
118.                this.type = type;
119.            }
120.            public double getMoney() {
121.                return money;
122.            }
123.            public void setMoney(double money) {
124.                this.money = money;
125.            }
```

```
126.
127.                public void topupMoney(double money){
128.                    this.money = this.money + money;
129.                }
130.
131.                public void returnMoney(double money){
132.                    this.money = this.money + money;
133.                }
134.
135.                public void payMoney(double money){
136.                    this.money = this.money - money;
137.                }
138.
139.                public boolean checkCanPay(double money){
140.                    boolean canPay = true;
141.                    if(this.money < money){
142.                        canPay = false;
143.                    }
144.                    return canPay;
145.                }
146.
147.                // ใช้เก็บ Promotion ที่เคยใช้แล้ว
148.                public List<Promotion> getPromotionList() {
149.                    return this.promotionList;
150.                }
151.
152.                public void setPromotionList(List<Promotion> promotionList) {
153.                    this.promotionList = promotionList;
154.                }
155.                public void addPromotion(Promotion promotion){
156.                    this.promotionList.add(promotion);
157.                }
158.                public void removePromotion(Promotion promotion){
159.                    this.promotionList.remove(promotion);
160.                }
161.
162.
163.                @Override
164.                public String toString() {
165.                    return ("User ID : "+ this.getId()+
166.                            "\nUsername : " + this.getUsername()+
167.                            "\nPassword : "+ this.getPassword()+
168.                            "\nFirstname : " + this.getFirstname()+
169.                            "\nLastname : " + this.getLastname()+
170.                            "\nEmail : " + this.getEmail()+
171.                            "\nType : " + this.getType()+
172.                            "\nMoney : " + this.getMoney()+
173.                            "\nPromotion : " + this.getPromotionList() + "\n"
174.                            );
175.                }
176.
177.            }
```

Class UserController

```
1.  package cinema;
2.
3.
4.      import java.util.ArrayList;
5.      import java.util.List;
6.      import javax.persistence.EntityManager;
7.      import javax.persistence.EntityManagerFactory;
8.      import javax.persistence.Persistence;
9.      import javax.persistence.Query;
10.
11.
12.     public class UserController {
13.         private UserController() {}
14.         private static UserController instance = new UserController();
15.         public static UserController getInstance() {
16.             return instance;
17.         }
18.
19.         public EntityManagerFactory emf;
20.         public EntityManager em;
21.         public void openConnection(){
22.             emf = Persistence.createEntityManagerFactory("db/CinemaODB.odb"); // co
    nnect to object database file
23.         em = emf.createEntityManager();
24.         }
25.         public void closeConnection(){
26.             em.close();
27.             emf.close();
28.         }
29.
30.         private List<User> userList = new ArrayList<User>();
31.
32.         // Operation
33.         public User getUser(int id){
34.             openConnection();
35.             em.getTransaction().begin(); // start connection
36.             // find object
37.             User user = em.find(User.class, id);
38.             for(Promotion p : user.getPromotionList()){} // get
39.             // Close the database connection:
40.             em.getTransaction().commit();
41.             closeConnection();
42.             return user;
43.         }
44.         public void addUser(User user){
45.
46.         openConnection();
47.
48.         em.getTransaction().begin(); // start connection
49.             em.persist(user); // add user to persist
50.             em.getTransaction().commit(); // add all persist to database
51.             closeConnection();
52.             //this.updateUserList(); // update list after add
53.         }
54.
55.         public void topupUserMoney(int id, double money){
56.             openConnection();
57.             User user = em.find(User.class, id); // find object
58.             em.getTransaction().begin();
59.             user.topupMoney(money);
60.             em.getTransaction().commit();
61.             closeConnection();
```

```
62.         }
63.
64.         public void editUser(int id, User editUser){
65.             openConnection();
66.             User user = em.find(User.class, id); // find object
67.             em.getTransaction().begin();
68.
69.             user.setUsername(editUser.getUsername());
70.             //user.setPassword(editUser.getPassword());
71.             user.setFirstname(editUser.getFirstname());
72.             user.setLastname(editUser.getLastname());
73.             user.setEmail(editUser.getEmail());
74.             user.setType(editUser.getType());
75.
76.             em.getTransaction().commit();
77.             closeConnection();
78.             updateUserList();
79.         }
80.
81.         public void editUserPassword(int id, String pass){
82.             openConnection();
83.             User user = em.find(User.class, id); // find object
84.             em.getTransaction().begin();
85.             user.setPassword(pass);
86.             em.getTransaction().commit();
87.             closeConnection();
88.         }
89.         public void deleteUser(int id){
90.             openConnection();
91.             em.getTransaction().begin(); // start connection
92.             Query query = em.createQuery("DELETE FROM User u WHERE u.id = :id");
93.
94.         query.setParameter("id", id).executeUpdate();
95.             em.getTransaction().commit(); // add all persist to database
96.             closeConnection();
97.             // update list after delete
98.             this.updateUserList();
99.         }
100.
101.                // Get all
102.                public void updateUserList() {
103.                    openConnection();
104.                    em.getTransaction().begin(); // start connection
105.                    Query q6 = em.createQuery("SELECT u FROM User u");
106.                    this.userList = q6.getResultList(); // get user
107.                    closeConnection();
108.                }
109.                public List<User> getUserList() {
110.                    updateUserList();
111.                    return userList;
112.                }
113.
114.                // Login & Logout
115.                private User loginUser;
116.                private boolean isLogin = false;
117.                public boolean checkValidUser(String username,String password){
118.                    updateUserList();
119.                    String pass = User.encryptPassword(password);
120.
121.                    for (User user : this.userList) {
122.                        if(user.getUsername().equals(username) && user.getPassword()
    .equals(pass)){
123.                            loginUser = user;
124.                            isLogin = true;
125.                            return true;
126.                        }
```

```
127.                     }
128.                     return false;
129.                 }
130.                 public boolean checkExistPassword(String username,String password){

131.                     updateUserList();
132.                     String pass = User.encryptPassword(password);
133.
134.                     for (User user : this.userList) {
135.                         if(user.getPassword().equals(pass)){
136.                             return true;
137.                         }
138.                     }
139.
140.                 return false;
141.                 }
142.                 public User getLoginUser() {
143.                     return loginUser;
144.                 }
145.                 public void setLogout(){
146.                     isLogin = false;
147.                 }
148.                 public boolean getIsLogin() {
149.                     return isLogin;
150.                 }
151.                 public void setIsLogin(boolean isLogin) {
152.                     this.isLogin = isLogin;
153.                 }
154.                 public void unsetLoginUser(){
155.                     this.loginUser = null;
156.                 }
157.
158.
159.                 // Register
160.                 public boolean checkExistEmail(String email){
161.                     this.updateUserList();
162.                         for (User u : this.userList) {
163.                             if (u.getEmail().equals(email)) {
164.                                 return true;
165.                             }
166.                         }
167.                     return false;
168.                 }
169.                 public boolean checkExistUsername(String username){
170.                     this.updateUserList();
171.                         for (User u : this.userList) {
172.                             if (u.getUsername().equals(username)) {
173.                                 return true;
174.                             }
175.                         }
176.                     return false;
177.                 }
178.
179.                 // Search
180.                 private List<User> userSearchList = new ArrayList<User>();
181.                 public void searchUser(String text, String searchOf){
182.                     openConnection();
183.                     em.getTransaction().begin(); // start connection
184.                     Query querySearch = em.createQuery("SELECT u FROM User u WHERE l
    ower(u." + searchOf +") LIKE lower(:" + searchOf +")", User.class);
185.                     querySearch.setParameter(searchOf.toString(), "%"+text+"%");
186.                     List<User> users = querySearch.getResultList(); // get user
187.                     this.userSearchList = users;
188.                     closeConnection();
189.                 }
190.                 public List<User> getUserSearchList(){
```

```
191.                     return this.userSearchList;
192.                 }
193.
194.                 // Booking
195.                 public void setUserMoney(int id, double money){
196.                     openConnection();
197.                     User user = em.find(User.class, id); // find object
198.                     em.getTransaction().begin();
199.                     user.setMoney(money);
200.                     em.getTransaction().commit();
201.                     closeConnection();
202.                 }
203.                 public Double getUserMoney(int id){
204.                     openConnection();
205.                     User user = em.find(User.class, id); // find object
206.                     closeConnection();
207.                     return user.getMoney();
208.                 }
209.
210.                 // Used Promotion
211.                 public void addUserPromotion(int id, Promotion promotion){
212.                     openConnection();
213.                     em.getTransaction().begin();
214.                     User user = em.find(User.class, id); // find object
215.                     user.addPromotion(promotion);
216.                     em.getTransaction().commit();
217.                     closeConnection();
218.                 }
219.
220.
221.
222.
223.
224.             }
```

Class CinemaController

```
1.  package cinema;
2.
3.
4.      import java.time.LocalDate;
5.      import java.time.format.DateTimeFormatter;
6.      import java.util.ArrayList;
7.      import java.util.List;
8.      import javax.persistence.EntityManager;
9.      import javax.persistence.EntityManagerFactory;
10.     import javax.persistence.Persistence;
11.     import javax.persistence.Query;
12.
13.
14.     public class CinemaController{
15.         private CinemaController() {}
16.         private static CinemaController instance = new CinemaController();
17.         public static CinemaController getInstance() {
18.             return instance;
19.         }
20.
21.         public EntityManagerFactory emf;
22.         public EntityManager em;
23.         public void openConnection(){
24.             emf = Persistence.createEntityManagerFactory("db/CinemaODB.odb");
25.  // connect to object database file
26.         em = emf.createEntityManager();
27.         }
28.         public void closeConnection(){
29.             em.close();
30.             emf.close();
31.         }
32.         // Select movie in main page any type of user
33.
34.         // Movie ================================================================
    =======
35.
36.         private Movie selectMovie;
37.         private boolean isComingSoon;
38.         public void setSelectMovie(int id){
39.             selectMovie = getMovie(id);
40.         }
41.         public Movie getSelectMovie(){
42.             return selectMovie;
43.
44.     }
45.         public boolean getIsComingSoon() {
46.             return isComingSoon;
47.         }
48.         public void setIsComingSoon(boolean isComingSoon) {
49.             this.isComingSoon = isComingSoon;
50.         }
51.
52.
53.         // Operation
54.         public Movie getMovie(int id){
55.             openConnection();
56.             // find object
57.             Movie movie = em.find(Movie.class, id);
58.             // Close the database connection:
59.             closeConnection();
60.             return movie;
61.         }
```

```
62.         public void addMovie(Movie movie){
63.             openConnection();
64.         em.getTransaction().begin(); // start connection
65.             em.persist(movie); // add user to persist
66.             em.getTransaction().commit(); // add all persist to database
67.             closeConnection();
68.         }
69.         public void editMovie(int id, Movie editMovie){
70.             openConnection();
71.             // find object
72.             Movie movie = em.find(Movie.class, id);
73.             em.getTransaction().begin();
74.             movie.setEnglishName(editMovie.getEnglishName());
75.             movie.setThaiName(editMovie.getThaiName());
76.             movie.setDirector(editMovie.getDirector());
77.             movie.setCast(editMovie.getCast());
78.             movie.setSynopsis(editMovie.getSynopsis());
79.             movie.setGenre(editMovie.getGenre());
80.             movie.setTime(editMovie.getTime());
81.             movie.setReleaseDate(editMovie.getReleaseDate());
82.             movie.setPoster(editMovie.getPoster());
83.             movie.setTrailer(editMovie.getTrailer());
84.             em.getTransaction().commit();
85.             // Close the database connection:
86.             closeConnection();
87.         }
88.         public void deleteMovie(int id){
89.             openConnection();
90.
91.         em.getTransaction().begin(); // start connection
92.             Query query = em.createQuery("DELETE FROM Movie m WHERE m.id = :id");
93.             query.setParameter("id", id).executeUpdate();
94.             em.getTransaction().commit(); // add all persist to database
95.             closeConnection();
96.         }
97.
98.         // Get all
99.         private List<Movie> movieList = new ArrayList<Movie>();
100.                public void updateMovieList() {
101.                    openConnection();
102.                    em.getTransaction().begin(); // start connection
103.                    Query query = em.createQuery("SELECT m FROM Movie m");
104.                    List<Movie> movies = query.getResultList(); // get movie
105.                    this.movieList = movies;
106.                    closeConnection();
107.                }
108.                public List<Movie> getMovieList() {
109.                    this.updateMovieList();
110.                    return movieList;
111.                }
112.
113.                // Search
114.                private List<Movie> movieSearchList = new ArrayList<Movie>();
115.                public void searchMovie(String text, String searchOf){
116.                    // ดึงข้อมูลจาก db ส่วน User
117.                    openConnection();
118.                    em.getTransaction().begin(); // start connection
119.                    Query querySearch = em.createQuery("SELECT m FROM Movie m WHERE
    lower(m." + searchOf +") LIKE lower(:" + searchOf +")", Movie.class);
120.                    querySearch.setParameter(searchOf.toString(), "%"+text+"%");
121.                    List<Movie> movies = querySearch.getResultList(); // get user
122.                    this.movieSearchList = movies;
123.                    closeConnection();
124.                }
125.                public List<Movie> getMovieSearchList(){
126.                    return this.movieSearchList;
```

```
127.                }
128.
129.                // Theatre ============================================================
    ================
130.                private List<Theatre> theatreList = new ArrayList<Theatre>(); // เก็บ
    โรง
131.
132.
133.                // Operation
134.                public Theatre getTheatre(int id){
135.                    openConnection();
136.
137.                em.getTransaction().begin(); // start connection
138.                    // find object
139.                    Theatre theatre = em.find(Theatre.class, id);
140.                    //for(Showtime st : theatre.getShowtimeList()){ } // get
141.                    em.getTransaction().commit();
142.                    closeConnection();
143.                    return theatre;
144.                }
145.                public void addTheatre(Theatre theatre){
146.                    openConnection();
147.                em.getTransaction().begin(); // start connection
148.                    em.persist(theatre); // add user to persist
149.                    em.getTransaction().commit(); // add all persist to database
150.                    closeConnection();
151.                }
152.                public void editTheatre(int id, Theatre editTheatre){
153.                    openConnection();
154.                    Theatre theatre = em.find(Theatre.class, id); // find object
155.                    em.getTransaction().begin();
156.                    theatre.setTheatreNumber(editTheatre.getTheatreNumber());
157.                    theatre.setTheatreType(editTheatre.getTheatreType());
158.                    em.getTransaction().commit();
159.                    closeConnection();
160.                }
161.                public void deleteTheatre(int id){
162.                    openConnection();
163.                    em.getTransaction().begin(); // start connection
164.                    Query query = em.createQuery("DELETE FROM Theatre t WHERE t.id =
    :id");
165.                    query.setParameter("id", id).executeUpdate();
166.                    em.getTransaction().commit(); // add all persist to database
167.                    closeConnection();
168.                }
169.
170.                // Get all
171.                public void updateTheatreList() {
172.                    openConnection();
173.                    em.getTransaction().begin(); // start connection
174.                    Query q6 = em.createQuery("SELECT t FROM Theatre t ORDER BY t.th
    eatreNumber ");
175.                    List<Theatre> theatres = q6.getResultList(); // get user
176.                    this.theatreList = theatres;
177.                    closeConnection();
178.                }
179.                public List<Theatre> getTheatreList() {
180.                    updateTheatreList();
181.
182.                return this.theatreList;
183.                }
184.
185.                // Add showtime
186.                public void addShowtime(int id, Showtime showtime){
187.                    openConnection();
```

```
188.                     em.getTransaction().begin();
189.                     Theatre theatre = em.find(Theatre.class, id); // find object
190.                     theatre.addShowtime(showtime);
191.                     em.getTransaction().commit();
192.                     closeConnection();
193.                 }
194.
195.                 public void deleteShowtime(int id,int idShowtime){
196.                     openConnection();
197.                     em.getTransaction().begin();
198.                     Theatre theatre = em.find(Theatre.class, id); // find object
199.                     Showtime showtime = em.find(Showtime.class, idShowtime);
200.                     for (Showtime st : theatre.getShowtimeList()) {
201.                         if(st.equals(showtime)){
202.                             theatre.deleteShowtime(st);
203.                         }
204.                     }
205.                     em.getTransaction().commit();
206.                     closeConnection();
207.                 }
208.
209.
210.                 // Showtime ============================================================
    ==================
211.                 private List<Showtime> showtimeList = new ArrayList<Showtime>(); //
    ก็บโรง
212.                 private int selectShowtime;
213.                 private boolean isSelectedSeat = false;
214.
215.
216.                 public boolean isSelectedSeat() {
217.                     return isSelectedSeat;
218.                 }
219.
220.
221.                 public void setIsSelectedSeat(boolean isSelectedSeat) {
222.                     this.isSelectedSeat = isSelectedSeat;
223.                 }
224.
225.                 public boolean checkShowtime(LocalDate date){
226.                     updateShowtimeList();
227.                     String formattedString = null;
228.                     try{
229.
230.                     LocalDate ld = date;
231.                         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d
     MMMM yyyy");
232.                         formattedString = ld.format(formatter);
233.                     }
234.                     catch(Exception e) {
235.                         System.out.println(e);
236.                     }
237.
238.                     for (Showtime showtime : showtimeList) {
239.                         if(showtime.getMovie().getId() == selectMovie.getId()){
240.                             if(showtime.getDate().equals(formattedString)){
241.                                 return true;
242.                             }
243.                         }
244.                     }
245.                     return false;
246.                 }
247.
248.                 public int getSelectShowtime(){
249.                     return this.selectShowtime;
```

```
250.                }
251.
252.                public void setSelectShowtime(int id){
253.                    this.selectShowtime = id;
254.                }
255.
256.
257.                // Operation
258.                public Showtime getShowtime(int id){
259.                    openConnection();
260.                    em.getTransaction().begin(); // start connection
261.                    // find object
262.                    Showtime showtime = em.find(Showtime.class, id);
263.            //        List <Seat> s = showtime.getSeatList();
264.
265.
266.                  //  for(Seat s : showtime.getSeatList()){} // get
267.                    // Close the database connection:
268.
269.
270.                    em.getTransaction().commit();
271.                    closeConnection();
272.                    return showtime;
273.                }
274.                public void addShowtime(Showtime showtime){
275.                    openConnection();
276.                em.getTransaction().begin(); // start connection
277.
278.                em.persist(showtime); // add user to persist
279.                    // add showtime into theatre list
280.                    em.flush();
281.                    Showtime st = em.find(Showtime.class, showtime.getId());
282.                    Theatre t = em.find(Theatre.class, st.getTheatre().getId());
283.                    t.addShowtime(st);
284.
285.                    em.getTransaction().commit(); // add all persist to database
286.                    closeConnection();
287.
288.                }
289.
290.                public void moveShowtime(int told,int tnew,int s){
291.                    //openConnection();
292.                   // em.getTransaction().begin();
293.
294.                    Showtime st = em.find(Showtime.class,s);
295.                    Theatre t1 = em.find(Theatre.class, told);
296.                    Theatre t2 = em.find(Theatre.class, tnew);
297.
298.                    t1.deleteShowtime(st);
299.                    t2.addShowtime(st);
300.
301.                  //  em.getTransaction().commit();
302.                  //  closeConnection();
303.                }
304.                public void removeTheatreShowtime(int t,int s){
305.                    Showtime st = em.find(Showtime.class,s);
306.                    Theatre t1 = em.find(Theatre.class, t);
307.
308.                    t1.deleteShowtime(st);
309.
310.                }
311.
312.                public void editShowtime(int id, Showtime editShowtime){
313.                    openConnection();
314.                    Showtime st = em.find(Showtime.class, id); // find object
315.                    em.getTransaction().begin();
```

```
316.
317.                     // แก้ Theatre Showtime List
318.                     if(st.getTheatre().getId() != editShowtime.getTheatre().getId())
    { // ถ้าเปลี่ยนโรง
319.                         moveShowtime(st.getTheatre().getId(),editShowtime.getTheatre
    ().getId(),id);
320.                         System.out.println("Change!");
321.                     }else{
322.
323.                     System.out.println("no change");
324.                     }
325.
326.                     // แก้
327.                     st.setMovie(editShowtime.getMovie());
328.                     st.setTheatre(editShowtime.getTheatre());
329.                     st.setSoundtrack(editShowtime.getSoundtrack());
330.                     st.setDate(editShowtime.getDate());
331.                     st.setStartTime(editShowtime.getOnlyStartTime());
332.             //         st.setDate(editShowtime.getDate());
333.                     st.setSystem(editShowtime.getSystem());
334.                     st.setIncreaseSeatPrice(editShowtime.getIncreaseSeatPrice());
335.                     // แก้ราคาใน seat ทั้งหมด
336.                     st.setSeatPrice(editShowtime.getIncreaseSeatPrice());
337.
338.                     em.getTransaction().commit();
339.                     closeConnection();
340.                 }
341.                 public void deleteShowtime(int id){
342.                     openConnection();
343.                     em.getTransaction().begin(); // start connection
344.                     Showtime st = em.find(Showtime.class, id); // find object
345.                     //st.getTheatre().deleteShowtime(st); // remove showtime
346.                     removeTheatreShowtime(st.getTheatre().getId(),id);
347.                     em.remove(st);
348.                     em.getTransaction().commit(); // add all persist to database
349.                     closeConnection();
350.                 }
351.
352.                 // Get all
353.                 public void updateShowtimeList() {
354.                     openConnection();
355.                     em.getTransaction().begin(); // start connection
356.                     Query q6 = em.createQuery("SELECT st FROM Showtime st");
357.                     List<Showtime> showtimes = q6.getResultList(); // get user
358.                     this.showtimeList = showtimes;
359.                     closeConnection();
360.                 }
361.                 public List<Showtime> getShowtimeList() {
362.                     updateShowtimeList();
363.                     return this.showtimeList;
364.                 }
365.
366.                 // Set seat book or unset
367.                 public void setShowtimeSeat(int id,String seatName,boolean seatStatu
    s){
368.                     openConnection();
369.
370.                 em.getTransaction().begin(); // start connection
371.                     // find object
372.                     Showtime showtime = em.find(Showtime.class, id);
373.                     showtime.setSeatStatus(seatName,seatStatus); // set book
374.                     // Close the database connection:
375.                     em.getTransaction().commit();
376.                     closeConnection();
377.                 }
```

```
378.
379.                // Get seat from showtime
380.                public Seat getShowtimeSeat(int id,String seatName){
381.                    openConnection();
382.                    em.getTransaction().begin(); // start connection
383.                    // find object
384.                    Showtime showtime = em.find(Showtime.class, id);
385.                    Seat seat = showtime.getSeat(seatName);
386.                    // Close the database connection:
387.                    em.getTransaction().commit();
388.                    closeConnection();
389.                    return seat;
390.                }
391.
392.                // Booking ================================================================
    ============================
393.                // Operation
394.                public Booking getBooking(int id){
395.                    openConnection();
396.                    em.getTransaction().begin(); // start connection
397.                    Booking booking = em.find(Booking.class, id);
398.                    em.getTransaction().commit();
399.                    closeConnection();
400.                    return booking;
401.                }
402.                public void addBooking(Booking booking){
403.                    openConnection();
404.                em.getTransaction().begin(); // start connection
405.                    em.persist(booking); // add user to persist
406.                    // add showtime into theatre list
407.                    em.flush();
408.                    Booking b = em.find(Booking.class, booking.getId()); // Get Last
     Add Booking
409.                    User u = em.find(User.class, b.getUser().getId()); // Get User
410.                    // หักเงิน
411.                    if(u.checkCanPay(booking.getTotalCost())){
412.                        u.payMoney(booking.getTotalCost());
413.
414.                    System.out.println("User pay okay");
415.                        if(b.getPromotion() != null){
416.                            Promotion p = em.find(Promotion.class, b.getPromotion().
    getPromotionID());
417.                            u.addPromotion(p);
418.                        }
419.                        System.out.println("Set Promotion to user okay");
420.                        // Set seat to booking true
421.                        List<Seat> seatList = b.getBookedSeatList();
422.                        for (Seat seat : seatList) {
423.                            Seat seatSet = em.find(Seat.class, seat.getId());
424.                            seatSet.setSeatStatus(true);
425.                        }
426.                        System.out.println("Set Seat alredy booked okay");
427.                        em.getTransaction().commit(); // add all persist to database

428.                        System.out.println("Book completed");
429.                    }else{
430.                        System.out.println("Money not enought to pay Please transfer
     money to account");
431.                    }
432.                    closeConnection();
433.                }
434.
435.                public void cancleBooking(int id){
436.                    openConnection();
437.                    em.getTransaction().begin(); // start connection
438.                    Booking b = em.find(Booking.class, id); // find object
```

```
439.                     User u = em.find(User.class, b.getUser().getId()); // Get User
440.
441.                     if(b.getPromotion() != null){
442.                         Promotion p = em.find(Promotion.class, b.getPromotion().getP
    romotionID());
443.                     }
444.
445.                     if(!b.isCancel()){
446.                         // u.removePromotion(p); // Remove Promotion from user
447.                         b.setIsCancel(true); // Set cancel to true
448.                         b.updateDatetime();
449.                         // Change money in user and booking
450.                         double userReturn  = b.cancelBooking(); // after cancel will
     set total only 10% and return user money
451.                         u.returnMoney(userReturn); // return to user account
452.
453.
454.                         // set ที่นั่งว่างเหมือนเดิม
455.                         List<Seat> seatList = b.getBookedSeatList();
456.                         for (Seat seat : seatList) {
457.
458.                         Seat seatSet = em.find(Seat.class, seat.getId());
459.                             seatSet.setSeatStatus(false);
460.                         }
461.                     }else{
462.                         System.out.println("Can't cancel again");
463.                     }
464.
465.
466.                     em.getTransaction().commit(); // add all persist to database
467.                     closeConnection();
468.                 }
469.
470.                 public void addStaffBooking(Booking booking){
471.                     openConnection();
472.                 em.getTransaction().begin(); // start connection
473.                     em.persist(booking); // add user to persist
474.                     // add showtime into theatre list
475.                     em.flush();
476.                     Booking b = em.find(Booking.class, booking.getId()); // Get Last
     Add Booking
477.
478.                     List<Seat> seatList = b.getBookedSeatList();
479.                         for (Seat seat : seatList) {
480.                             Seat seatSet = em.find(Seat.class, seat.getId());
481.                             seatSet.setSeatStatus(true);
482.                         }
483.                         System.out.println("Set Seat alredy booked okay");
484.                         em.getTransaction().commit(); // add all persist to database

485.                         System.out.println("Book completed");
486.                     closeConnection();
487.                 }
488.                 public void cancleStaffBooking(int id){
489.                     openConnection();
490.                     em.getTransaction().begin(); // start connection
491.                     Booking b = em.find(Booking.class, id); // find object
492.                     if(!b.isCancel()){
493.                         // u.removePromotion(p); // Remove Promotion from user
494.                         b.setIsCancel(true); // Set cancel to true
495.                         b.updateDatetime();
496.                         // Change money in user and booking
497.                         double userReturn  = b.cancelBooking(); // after cancel will
     set total only 10% and return user money
498.
499.
```

```
500.                        // set ที่นั่งว่างเหมือนเดิม
501.                        List<Seat> seatList = b.getBookedSeatList();
502.                        for (Seat seat : seatList) {
503.
504.                        Seat seatSet = em.find(Seat.class, seat.getId());
505.                            seatSet.setSeatStatus(false);
506.                        }
507.                    }else{
508.                        System.out.println("Can't cancel again");
509.                    }
510.                    em.getTransaction().commit(); // add all persist to database
511.                    closeConnection();
512.                }
513.
514.                // Get all
515.                private List<Booking> bookingList = new ArrayList<Booking>();
516.                public void updateBookingList() {
517.                    openConnection();
518.                    em.getTransaction().begin(); // start connection
519.                    Query query = em.createQuery("SELECT b FROM Booking b");
520.                    List<Booking> booking = query.getResultList(); // get movie
521.                    this.bookingList = booking;
522.                    closeConnection();
523.                }
524.                public List<Booking> getBookingList() {
525.                    this.updateBookingList();
526.                    return bookingList;
527.                }
528.                public List<Booking> getMyBookingList(int userId) {
529.                    this.updateBookingList();
530.                    List<Booking> userBookingList = new ArrayList<Booking>();
531.                    for (Booking b : this.bookingList) {
532.                        if(b.getUser().getId() == userId){
533.                            userBookingList.add(b);
534.                        }
535.                    }
536.                    return userBookingList;
537.                }
538.                public double getSumTotal(){
539.                    double total = 0;
540.                    this.updateBookingList();
541.                    for (Booking booking : bookingList) {
542.                        total += booking.getTotalCost();
543.                    }
544.                    return total;
545.                }
546.
547.
548.
549.                List<Seat> seatList = new ArrayList<Seat>();
550.                List<String> selectSeatName = new ArrayList<String>();
551.                public void setSeatList(List<String> selectSeat){
552.                    selectSeatName.clear();
553.                    selectSeatName = selectSeat;
554.                } // ใส่ iv เข้ามาเป็น string
555.                public List<Seat> getSeatList(){
556.                    seatList.clear(); // clear before get new seat
557.                    for (String seatString : selectSeatName) {
558.                        seatList.add(getShowtimeSeat(selectShowtime,seatString));
559.                    }
560.                    return seatList;
561.                } // get ออกไปเป็น seat
562.            }
```

Class Main

```java
package cinema;


    import java.util.ArrayList;
    import java.util.List;


    public class Main{
        public static void main(String[] args) {
           CinemaController cc = CinemaController.getInstance();
           UserController uc = UserController.getInstance();
           PromotionController pc = PromotionController.getInstance();

           Showtime s = cc.getShowtime(9);

           List<Seat> selectSeat = new ArrayList<Seat>();
           // จำลองดึง Seat มา 3 ตัว
           Seat s1 = cc.getShowtimeSeat(s.getId(),"A1");
           Seat s2 = cc.getShowtimeSeat(s.getId(),"F20");
           Seat s3 = cc.getShowtimeSeat(s.getId(),"B20");

           selectSeat.add(s1);
           selectSeat.add(s2);
           selectSeat.add(s3);

           User u = uc.getUser(4);
           Promotion p = pc.getPromotion(9);

           // get Total seat Price
           double totalSeatPrice = 0;
            for (Seat seat : selectSeat) {
                totalSeatPrice += seat.getSeatPrice();
            }
           // System.out.println(totalSeatPrice);
           // Check promotion and make total
           double totalPrice = totalSeatPrice;
           if(p != null){
              // selected promotion
              totalPrice -= p.getDiscount();
           }

           // System.out.println(totalPrice);

           Booking b = new Booking(s, selectSeat, u, p, totalPrice);
           //System.out.println(b);

           pc.getUnusePromotion(15);

        }
    }
```

ส่วนadmin

Class AdminMainController

```
1.      package cinema.admin;
2.
3.
4.      import cinema.Booking;
5.      import cinema.CinemaController;
6.      import cinema.Movie;
7.      import cinema.Promotion;
8.      import cinema.PromotionController;
9.      import cinema.Showtime;
10.     import cinema.Theatre;
11.     import cinema.UserController;
12.     import cinema.User;
13.     import cinema.ui.AlertMaker;
14.     import com.jfoenix.controls.JFXTabPane;
15.     import java.io.File;
16.     import java.io.IOException;
17.     import java.io.InputStream;
18.     import java.net.URL;
19.     import java.text.DecimalFormat;
20.     import java.text.ParseException;
21.     import java.time.LocalDate;
22.     import java.time.format.DateTimeFormatter;
23.     import java.util.ArrayList;
24.     import java.util.List;
25.     import java.util.Optional;
26.     import java.util.ResourceBundle;
27.     import java.util.regex.Matcher;
28.     import java.util.regex.Pattern;
29.     import javafx.collections.FXCollections;
30.     import javafx.collections.ObservableList;
31.     import javafx.event.ActionEvent;
32.     import javafx.fxml.FXML;
33.     import javafx.fxml.FXMLLoader;
34.     import javafx.fxml.Initializable;
35.     import javafx.scene.Parent;
36.     import javafx.scene.Scene;
37.     import javafx.scene.control.Alert;
38.     import javafx.scene.control.Button;
39.     import javafx.scene.control.ButtonType;
40.     import javafx.scene.control.ChoiceBox;
41.     import javafx.scene.control.ComboBox;
42.
43. import javafx.scene.control.DatePicker;
44.     import javafx.scene.control.Label;
45.     import javafx.scene.control.Menu;
46.     import javafx.scene.control.MenuBar;
47.     import javafx.scene.control.MenuItem;
48.     import javafx.scene.control.PasswordField;
49.     import javafx.scene.control.SingleSelectionModel;
50.     import javafx.scene.control.Tab;
51.     import javafx.scene.control.TableColumn;
52.     import javafx.scene.control.TableView;
53.     import javafx.scene.control.TextArea;
54.     import javafx.scene.control.TextField;
55.     import javafx.scene.control.cell.PropertyValueFactory;
56.     import javafx.scene.image.Image;
57.     import javafx.scene.image.ImageView;
58.     import javafx.scene.input.MouseEvent;
59.     import javafx.scene.paint.Color;
```

```
60.     import javafx.scene.text.Text;
61.     import javafx.stage.FileChooser;
62.     import javafx.stage.Stage;
63.     import javafx.stage.StageStyle;
64.
65.
66.
67.
68.     public class AdminMainController implements Initializable{
69.         // User
70.         @FXML
71.         private TextField txtUserSearch;
72.         @FXML
73.         private Button btnCancleUserSearch;
74.         @FXML
75.         private Label lbHeadUser;
76.         @FXML
77.         private Label lbUsername;
78.         @FXML
79.         private TextField txtUsername;
80.         @FXML
81.         private Label lbPassword;
82.         @FXML
83.         private PasswordField txtPassword;
84.         @FXML
85.         private Label lbConfirmPassword;
86.         @FXML
87.         private PasswordField txtConfirmPassword;
88.         @FXML
89.         private Label lbFirstname;
90.         @FXML
91.
92.     private TextField txtFirstname;
93.         @FXML
94.         private Label lbLastname;
95.         @FXML
96.         private TextField txtLastname;
97.         @FXML
98.         private Label lbEmail;
99.         @FXML
100.                private TextField txtEmail;
101.                @FXML
102.                private Button btnAddUser;
103.                @FXML
104.                private Button btnSaveUser;
105.                @FXML
106.                private Button btnCancleEditUser;
107.                @FXML
108.                private TableView<User> tbUser;
109.                @FXML
110.                private TableColumn<User, Integer> colUserId;
111.                @FXML
112.                private TableColumn<User, String> colUsername;
113.                @FXML
114.                private TableColumn<User, String> colPassword;
115.                @FXML
116.                private TableColumn<User, String> colFirstname;
117.                @FXML
118.                private TableColumn<User, String> colLastname;
119.                @FXML
120.                private TableColumn<User, String> colEmail;
121.                @FXML
122.                private TableColumn<User, String> colType;
123.                @FXML
124.                private Label lbUserSearch;
125.                @FXML
```

```
126.                private ChoiceBox<String> cbUserType;
127.                @FXML
128.                private Label msgUsername;
129.                @FXML
130.                private Label msgPassword;
131.                @FXML
132.                private Label msgConfirmPassword;
133.                @FXML
134.                private Label msgFirstname;
135.                @FXML
136.                private Label msgLastname;
137.                @FXML
138.
139.            private Label msgEmail;
140.                @FXML
141.                private Label lbUserType;
142.                @FXML
143.                private Label msgUserType;
144.                @FXML
145.                private Label msgUserSearch;
146.                @FXML
147.                private ComboBox<String> cbUserSearch;
148.                @FXML
149.                private Button btnUserPassword;
150.
151.                // ============================================================
152.                // Movie
153.                @FXML
154.                private Label lbMovieSearch;
155.                @FXML
156.                private TextField txtMovieSearch;
157.                @FXML
158.                private Label msgMovieSearch;
159.                @FXML
160.                private ComboBox<String> cbMovieSearch;
161.                @FXML
162.                private Button btnCancleMovieSearch;
163.                @FXML
164.                private Label lbHeadMovie;
165.                @FXML
166.                private Label lbEnglishName;
167.                @FXML
168.                private TextField txtEnglishName;
169.                @FXML
170.                private Label lbThaiName;
171.                @FXML
172.                private TextField txtThaiName;
173.                @FXML
174.                private Label msgThaiName;
175.                @FXML
176.                private Label lbDirector;
177.                @FXML
178.                private TextField txtDirector;
179.                @FXML
180.                private Label msgDirector;
181.                @FXML
182.                private Label lbCast;
183.                @FXML
184.                private TextField txtCast;
185.
186.            @FXML
187.                private Label msgCast;
188.                @FXML
189.                private Label lbSynopsis;
190.                @FXML
191.                private TextArea txtSynopsis;
```

```
192.                @FXML
193.                private Label msgSynopsis;
194.                @FXML
195.                private Label lbGenre;
196.                @FXML
197.                private Label msgGenre;
198.                @FXML
199.                private Label lbTime;
200.                @FXML
201.                private Label lbTimeHour;
202.                @FXML
203.                private ChoiceBox<String> cbTimeHour;
204.                @FXML
205.                private Label lbTimeMinute;
206.                @FXML
207.                private ChoiceBox<String> cbTimeMinute;
208.                private Label msgTime;
209.                @FXML
210.                private Label lbReleaseDate;
211.                @FXML
212.                private DatePicker datePickerReleaseDate;
213.                @FXML
214.                private Label msgReleaseDate;
215.                @FXML
216.                private Label lbPoster;
217.                @FXML
218.                private Button filePoster;
219.                @FXML
220.                private ImageView imagePoster;
221.                @FXML
222.                private Label msgPoster;
223.                @FXML
224.                private Label lbTrailer;
225.                @FXML
226.                private TextField txtTrailer;
227.                @FXML
228.                private Label msgTrailer;
229.                @FXML
230.                private Button btnAddMovie;
231.                @FXML
232.
233.            private Button btnSaveMovie;
234.                @FXML
235.                private Button btnCancleEditMovie;
236.                @FXML
237.                private TableView<Movie> tbMovie;
238.                @FXML
239.                private TableColumn<Movie, Integer> colMovieId;
240.                @FXML
241.                private TableColumn<Movie, String> colEnglishName;
242.                @FXML
243.                private TableColumn<Movie, String> colThaiName;
244.                @FXML
245.                private TableColumn<Movie, String> colDirector;
246.                @FXML
247.                private TableColumn<Movie, String> colCast;
248.                @FXML
249.                private TableColumn<Movie, String> colSynopsis;
250.                @FXML
251.                private TableColumn<Movie, String> colGenre;
252.                @FXML
253.                private TableColumn<Movie, String> colTime;
254.                @FXML
255.                private TableColumn<Movie, String> colReleaseDate;
256.                @FXML
257.                private TableColumn<Movie, String> colPoster;
```

```
258.                @FXML
259.                private TableColumn<Movie, String> colTrailer;
260.                @FXML
261.                private Label msgEnglishName;
262.                @FXML
263.                private ChoiceBox<String> cbGenre;
264.                @FXML
265.                private JFXTabPane tabPane;
266.                @FXML
267.                private Label msgHour;
268.                @FXML
269.                private Label msgMinute;
270.                @FXML
271.                private TableColumn<User, Double> colMoney;
272.                // ================================================================
    ==========
273.                // Promotion
274.
275.                @FXML
276.                private Label lbPromotionSearch;
277.
278.            @FXML
279.                private TextField txtPromotionSearch;
280.                @FXML
281.                private Label msgPromotionSearch;
282.                @FXML
283.                private Button btnCancelPromotionSearch;
284.                @FXML
285.                private Label lbHeadPromotion;
286.                @FXML
287.                private Label lbPromotion;
288.                @FXML
289.                private TextField txtPromotion;
290.                @FXML
291.                private Label msgPromotion;
292.                @FXML
293.                private Label lbDescription;
294.                @FXML
295.                private TextField txtDescription;
296.                @FXML
297.                private Label msgDescription;
298.                @FXML
299.                private Label lbDiscount;
300.                @FXML
301.                private TextField txtDiscount;
302.                @FXML
303.                private Label msgDiscount;
304.                @FXML
305.                private Label lbDate;
306.                @FXML
307.                private DatePicker pickerDate;
308.                @FXML
309.                private Label msgDate;
310.                @FXML
311.                private Button btnAddPromotion;
312.                @FXML
313.                private Button btnSavePromotion;
314.                @FXML
315.                private Button btnCancelPromotion;
316.                @FXML
317.                private TableView<Promotion> tbPromotion;
318.                @FXML
319.                private MenuItem handleUserDelete1;
320.                @FXML
321.                private TableColumn<Promotion, Integer> colPromotionId;
322.                @FXML
```

```
323.                private TableColumn<Promotion, String> colPromotionName;
324.
325.            @FXML
326.                private TableColumn<Promotion, String> colDate;
327.                @FXML
328.                private TableColumn<Promotion, String> colDescription;
329.                @FXML
330.                private TableColumn<Promotion, Double> colDiscount;
331.
332.                // Cinema login
333.                // ================================================================
    ======
334.                @FXML
335.                private Menu userLogin;
336.                // ================================================================
    ==========
337.                // Theatre
338.                @FXML
339.                private TextField txtTheatre;
340.                @FXML
341.                private Label msgTheatre;
342.                @FXML
343.                private Button btnAddTheatre;
344.                @FXML
345.                private Button btnSaveTheatre;
346.                @FXML
347.                private Button btnCancelTheatre;
348.                @FXML
349.                private TableView<Theatre> tbTheatre;
350.                @FXML
351.                private TableColumn<Theatre, Integer> colTheatreId;
352.                @FXML
353.                private TableColumn<Theatre, Integer> colTheatreNumber;
354.                @FXML
355.                private Label lbHeadTheatre;
356.
357.                // Showtime
358.                // ================================================================
    ==============
359.                @FXML
360.                private Label lbHeadShowtime1;
361.                @FXML
362.                private Label lbMovieSearch1;
363.                @FXML
364.                private ComboBox<String> cbShowtimeSearch1;
365.                @FXML
366.
367.            private Label lbTheatreShow1;
368.                @FXML
369.                private ComboBox<String> cbTheatreSearch1;
370.                @FXML
371.                private Label lbSoundtrack1;
372.                @FXML
373.                private ComboBox<String> cbSoundtrack1;
374.                @FXML
375.                private Label lbIncreaseSeatPrice1;
376.                @FXML
377.                private TextField txtIncreaseSeatPrice1;
378.                @FXML
379.                private Label msgIncreaseSeatPrice1;
380.                @FXML
381.                private Label lbStartTime1;
382.                @FXML
383.                private Label lbTimeHourShow1;
384.                @FXML
385.                private ChoiceBox<String> cbTimeHourShow1;
```

```
386.                @FXML
387.                private Label lbTimeMinuteShow1;
388.                @FXML
389.                private ChoiceBox<String> cbTimeMinuteShow1;
390.                @FXML
391.                private Button btnAddShowtime1;
392.                @FXML
393.                private Button btnSaveShowtime1;
394.                @FXML
395.                private Button btnCancleEditShowtime1;
396.                @FXML
397.                private TableView<Showtime> tbShowtime1;
398.                @FXML
399.                private TableColumn<Showtime, Integer> colShowtimeId1;
400.                @FXML
401.                private TableColumn<Showtime, Movie> colMoviesName1;
402.                @FXML
403.                private TableColumn<Showtime, Theatre> colTheatreShow1;
404.                @FXML
405.                private TableColumn<Showtime, String> colSoundtrackShow1;
406.                @FXML
407.                private TableColumn<Showtime, String> colPeriodShow1;
408.                @FXML
409.                private TableColumn<Showtime, Integer> colIncreaseSeatPrice1;
410.                @FXML
411.                private Label msgMovieSearch1;
412.                @FXML
413.
414.            private Label msgTheatreShow1;
415.                @FXML
416.                private Label msgTimeShowtime;
417.                @FXML
418.                private Label lbSoundtrack11;
419.                @FXML
420.                private ComboBox<String> cbSystem;
421.                @FXML
422.                private TableColumn<Showtime, String> colShowtimeSystem;
423.                @FXML
424.                private TableColumn<Showtime, String> colSubtitle;
425.                // Showtime Date
426.                @FXML
427.                private DatePicker datePickerShowtime;
428.                @FXML
429.                private Label msgShowtimeDate;
430.                @FXML
431.                private TableColumn<Showtime, String> colShowtimeDate;
432.                @FXML
433.                private ComboBox<String> cbTheatreType;
434.                @FXML
435.                private TableColumn<Theatre, String> colTheatreType;
436.                // Booking
437.                // ========================================================
438.                @FXML
439.                private Text txtSummary;
440.                @FXML
441.                private TableView<Booking> tbBooking;
442.                @FXML
443.                private TableColumn<Booking, Integer> colBookingID;
444.                @FXML
445.                private TableColumn<Booking, String> colBookingCreated;
446.                @FXML
447.                private TableColumn<Booking, String> colBookingUpdate;
448.                @FXML
449.                private TableColumn<Booking, String> colBookingShowtime;
450.                @FXML
451.                private TableColumn<Booking, String> colBookingUser;
```

```
452.                @FXML
453.                private TableColumn<Booking, Promotion> colBookingPromotion;
454.                @FXML
455.                private TableColumn<Booking, Boolean> colBookingStatus;
456.                @FXML
457.                private TableColumn<Booking, String> colBookingSeat;
458.                @FXML
459.                private TableColumn<Booking, Double> colBookingTotal;
460.
461.
462.
463.                // User ============================================================
    ===========
464.                // User normal
465.                ObservableList<User> userList = FXCollections.observableArrayList();

466.                List<User> userListArray = new ArrayList<User>();
467.                // User search
468.                ObservableList<User> userSearchList = FXCollections.observableArrayL
    ist();
469.                List<User> userSearchListArray = new ArrayList<User>();
470.                // Movie ===========================================================
    ===========
471.                // Movie normal
472.                ObservableList<Movie> movieList = FXCollections.observableArrayList(
    );
473.                List<Movie> movieListArray = new ArrayList<Movie>();
474.                // Movie search
475.                ObservableList<Movie> movieSearchList = FXCollections.observableArra
    yList();
476.                List<Movie> movieSearchListArray = new ArrayList<Movie>();
477.                // Promotion =======================================================
    ===============
478.                // Promotion Normal
479.                ObservableList<Promotion> promotionList = FXCollections.observableAr
    rayList();
480.                List<Promotion> promotionListArray = new ArrayList<Promotion>();
481.                // Promotion Search
482.                ObservableList<Promotion> promotionSearchList = FXCollections.observ
    ableArrayList();
483.                List<Promotion> promotionSearchListArray = new ArrayList<Promotion>(
    );
484.                // Theatre =========================================================
    =============
485.                // Theatre Normal
486.                ObservableList<Theatre> theatreList = FXCollections.observableArrayL
    ist();
487.                List<Theatre> theatreListArray = new ArrayList<Theatre>();
488.                // Showtime ========================================================
    ===========
489.                // Showtime normal
490.                ObservableList<Showtime> showtimeList = FXCollections.observableArra
    yList();
491.                List<Showtime> showtimeListArray = new ArrayList<Showtime>();
492.                // Showtime search
493.                ObservableList<Showtime> showtimeSearchList = FXCollections.observab
    leArrayList();
494.                List<Showtime> showtimeSearchListArray = new ArrayList<Showtime>();

495.
496.            // Booking List ================================================
    ============
497.                ObservableList<Booking> bookingList = FXCollections.observableArrayL
    ist();
498.                List<Booking> bookingListArray = new ArrayList<Booking>();
499.
```

```
500.                private File lastPath;
501.                private File selectedPoster;
502.                private String fileName = "";
503.                private Image posterImage;
504.
505.
506.                private UserController uc;
507.                private PromotionController pc;
508.                private CinemaController cc;
509.                @FXML
510.                private MenuBar menuAdmin;
511.
512.
513.                public AdminMainController() {
514.                    this.uc = uc.getInstance();
515.                    this.pc = pc.getInstance();
516.                    this.cc = cc.getInstance();
517.                }
518.
519.                // Initial ============================================================
    ============
520.                @Override
521.                public void initialize(URL url, ResourceBundle rb) {
522.                    if(uc.getIsLogin()){
523.                        userLogin.setText(uc.getLoginUser().getEmail());
524.                    }else{
525.                        userLogin.setText("Not Login");
526.                        userLogin.setDisable(true);
527.                    }
528.
529.                    // Tab select
530.                    SingleSelectionModel<Tab> selectionModel = tabPane.getSelectionM
    odel();
531.                    selectionModel.select(4); // select theatre
532.                    // User ===============================================================
    =
533.                    // User load data
534.                    initUserCol();
535.                    loadUserData();
536.                    // Clear Message text
537.                    clearUserMessage();
538.
539.                // User combobox type load
540.                    loadUserCombo();
541.                    // Clear user search message
542.                    clearUserSearchMessage();
543.                    // User search combo
544.                    loadUserSearchCombo();
545.                    // Disable cancle user search
546.                    btnCancleUserSearch.setDisable(true);
547.
548.                    // Movie ==============================================================
    ==
549.                    initMovieCol();
550.                    loadMovieData();
551.                    clearMovieMessage();
552.                    // Load combo
553.                    loadMovieSearchCombo(); // search combo
554.                    loadMovieGenreCombo(); // genre combo
555.                    loadMovieHourCombo(); // hour combo
556.                    loadMovieMinuteCombo(); // minute combo
557.                    btnCancleMovieSearch.setDisable(true);
558.                    clearMovieSearchMessage();
559.
560.                    // Promotion ----------------------------------------------
561.                    initPromotionCol();
```

```
562.                    loadPromotionData();
563.                    clearPromotionMessage();
564.                    btnCancelPromotionSearch.setDisable(true);
565.                    clearPromotionSearchMessage();
566.
567.
568.                    // Theatre
569.                    initTheatreCol();
570.                    loadTheatreData();
571.                    clearTheatreMessage();
572.                    loadTheatreTypeCombo();
573.
574.                    // Showtime ============================================================
    ==
575.                    initShowtimeCol();
576.                    loadShowtimeData();
577.                    clearShowtimeMessage();
578.            //        // Load combo
579.                    loadShowtimeSearchMovieCombo(); // movie
580.                    loadTheatreShowtimeCombo(); // theatre
581.                    loadSoundtrackCombo(); // soundtrack
582.                    loadStartHourCombo();
583.                    loadStartMinCombo();
584.                    cbSystem.setDisable(true);
585.                    //loadSystemCombo();
586.
587.
588.
589.            //        Theatre t = tc.getTheatre(3);
590.            //        Movie m = mc.getMovie(4);
591.            //        Showtime s = new Showtime(m, t, "3D", "TH", "11:11", 0.0);
592.            //        sc.addShowtime(s);
593.                    initBookingCol();
594.                    loadBookingData();
595.                    setBookingTotal();
596.                }
597.
598.                @FXML
599.                private void handleLogout(ActionEvent event) throws IOException {
600.                    Parent parent;
601.                    parent = FXMLLoader.load(getClass().getResource("/cinema/ui/auth
    /Login.fxml"));
602.                    Scene parentScene = new Scene(parent);
603.                    Stage window = (Stage)menuAdmin.getScene().getWindow();
604.                    window.setScene(parentScene);
605.                    window.show();
606.
607.
608.                    uc.setIsLogin(false);
609.                    uc.unsetLoginUser();
610.
611.
612.
613.            System.out.println("Logout");
614.                }
615.
616.                public boolean checkEmptyForm(String text, Label label, String textL
    abel){
617.                    if (text.equals("")) {
618.                        label.setText("Your " + textLabel + " is empty!");
619.                        label.setTextFill(Color.rgb(210, 39, 30));
620.                    } else {
621.                        label.setText("Your " + textLabel + " is okay.");
622.                        label.setTextFill(Color.rgb(21, 117, 84));
623.                        return true;
624.                    }
```

```
625.                     return false;
626.                 }
627.
628.                 // User ============================================================
    ============================
629.                 public void initUserCol(){
630.                     colUserId.setCellValueFactory(new PropertyValueFactory<>("id"));

631.
632.                 colUsername.setCellValueFactory(new PropertyValueFactory<>("username
    "));
633.                     colPassword.setCellValueFactory(new PropertyValueFactory<>("pass
    word"));
634.                     colFirstname.setCellValueFactory(new PropertyValueFactory<>("fir
    stname"));
635.                     colLastname.setCellValueFactory(new PropertyValueFactory<>("last
    name"));
636.                     colEmail.setCellValueFactory(new PropertyValueFactory<>("email")
    );
637.                     colType.setCellValueFactory(new PropertyValueFactory<>("type"));

638.                     colMoney.setCellValueFactory(new PropertyValueFactory<>("money")
    );
639.                 }
640.                 public void loadUserData(){
641.                     userList.clear();
642.                     userListArray = uc.getUserList();
643.                     try {
644.                         for (User user : userListArray) {
645.                             userList.add(user);
646.                         }
647.                     } catch (Exception e) {
648.                         System.out.println(e);
649.                     }
650.
651.                     tbUser.setItems(userList);
652.
653.                 }
654.                 public void loadUserCombo(){
655.                     ObservableList<String> userTypeOptions = FXCollections.observabl
    eArrayList("Customer","Staff","Admin");
656.                     cbUserType.setItems(userTypeOptions);
657.                     //cbUserType.setValue("Customer");
658.                 }
659.                 public void loadUserSearchCombo(){
660.                     ObservableList<String> userSearchOptions = FXCollections.observa
    bleArrayList("Username","Firstname","Lastname","Email","Type");
661.                     cbUserSearch.setItems(userSearchOptions);
662.                     cbUserSearch.setValue("Username");
663.                 }
664.                 public void clearUserMessage(){
665.                     msgUsername.setText(null);
666.                     msgPassword.setText(null);
667.                     msgConfirmPassword.setText(null);
668.                     msgFirstname.setText(null);
669.                     msgLastname.setText(null);
670.                     msgEmail.setText(null);
671.
672.                 msgUserType.setText(null);
673.                 }
674.                 public void clearUserSearchMessage(){
675.                     msgUserSearch.setText(null);
676.                 }
677.                 public void clearUserForm(){
678.                     txtUsername.clear();
679.                     txtPassword.clear();
```

```
680.                    txtConfirmPassword.clear();
681.                    txtFirstname.clear();
682.                    txtLastname.clear();
683.                    txtEmail.clear();
684.                    cbUserType.setValue(null);
685.                }
686.                public void cancleUserEdit(){
687.                    btnAddUser.setDisable(false);
688.                    btnSaveUser.setDisable(true);
689.                    btnCancleEditUser.setDisable(true);
690.                    lbHeadUser.setText("Add new user");
691.                    clearUserForm();
692.                    clearUserMessage();
693.                    editUserMode = false;
694.                    btnUserPassword.setDisable(true);
695.
696.                    txtUsername.setDisable(false);
697.                    txtPassword.setDisable(false);
698.                    txtPassword.setDisable(false);
699.                    txtConfirmPassword.setDisable(false);
700.                    txtFirstname.setDisable(false);
701.                    txtLastname.setDisable(false);
702.                    txtEmail.setDisable(false);
703.                    cbUserType.setDisable(false);
704.                    editUserPassword = false;
705.                }
706.
707.                User selectedUserEdit; // global variable
708.                boolean editUserMode = false;
709.                int editUserId = 0;
710.
711.                public void initUserEditMode(){
712.                    selectedUserEdit = tbUser.getSelectionModel().getSelectedItem();
    // เก็บมาเป็น object จาก list ที่เลือก
713.                    if(selectedUserEdit == null){
714.                        AlertMaker.showErrorMessage("No user selected", "Please sele
    ct a user for edit.");
715.                        return;
716.
717.                }
718.                    txtUsername.setDisable(false);
719.                    txtPassword.setDisable(false);
720.                    txtPassword.setDisable(true);
721.                    txtConfirmPassword.setDisable(true);
722.                    txtFirstname.setDisable(false);
723.                    txtLastname.setDisable(false);
724.                    cbUserType.setDisable(false);
725.
726.
727.                    // Disable button to add & Enable button to save & Change head t
    ext to edit mode
728.                    btnAddUser.setDisable(true);
729.                    btnSaveUser.setDisable(false);
730.                    btnCancleEditUser.setDisable(false);
731.                    lbHeadUser.setText("Edit user id : " + selectedUserEdit.getId())
    ;
732.
733.                    txtUsername.setText(selectedUserEdit.getUsername());
734.                    txtPassword.setText(selectedUserEdit.getPassword());
735.                    // Disable txt Password
736.                    txtPassword.setDisable(true);
737.                    txtConfirmPassword.setDisable(true);
738.                    btnUserPassword.setDisable(false);
739.
740.                    txtConfirmPassword.setText(selectedUserEdit.getPassword());
741.                    txtFirstname.setText(selectedUserEdit.getFirstname());
```

```
742.                     txtLastname.setText(selectedUserEdit.getLastname());
743.                     txtEmail.setText(selectedUserEdit.getEmail());
744.                     cbUserType.setValue(selectedUserEdit.getType());
745.                     clearUserMessage();
746.                     editUserMode = true;
747.                     editUserId = tbUser.getSelectionModel().getSelectedIndex();
748.                 }
749.                 public boolean checkExistUsername(String username, Label label, Stri
    ng textLabel){
750.                     if (uc.checkExistUsername(username)) {
751.                         label.setText("Your " + textLabel + " is already used!");
752.                         label.setTextFill(Color.rgb(210, 39, 30));
753.                     } else {
754.                         label.setText("Your " + textLabel + " is okay.");
755.                         label.setTextFill(Color.rgb(21, 117, 84));
756.                         return true;
757.                     }
758.                     return false;
759.                 }
760.
761.             public boolean checkExistEmail(String email, Label label, String textLab
    el){
762.                     if (uc.checkExistEmail(email)) {
763.                         label.setText("Your " + textLabel + " is already used!");
764.                         label.setTextFill(Color.rgb(210, 39, 30));
765.                     } else {
766.                         label.setText("Your " + textLabel + " is okay.");
767.                         label.setTextFill(Color.rgb(21, 117, 84));
768.                         return true;
769.                     }
770.                     return false;
771.                 }
772.                 public boolean checkPasswordMatch(String password, String confirmPas
    sword, Label label, String textLabel){
773.                     if (!password.equals(confirmPassword)) {
774.                         label.setText("Your " + textLabel + " is not match!");
775.                         label.setTextFill(Color.rgb(210, 39, 30));
776.                     } else {
777.                         label.setText("Your " + textLabel + " is okay.");
778.                         label.setTextFill(Color.rgb(21, 117, 84));
779.                         return true;
780.                     }
781.                     return false;
782.                 }
783.                 public static boolean isValidEmail(String email, Label label, String
     textLabel) {
784.                     String ePattern = "^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@((\\[[0-
    9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\])|(([a-zA-Z\\-0-9]+\\.)+[a-zA-
    Z]{2,}))$";
785.                     Pattern p = java.util.regex.Pattern.compile(ePattern);
786.                     Matcher m = p.matcher(email);
787.                     if(!m.matches()){
788.                         label.setText("Your " + textLabel + " is wrong format!");
789.                         label.setTextFill(Color.rgb(210, 39, 30));
790.                     }else{
791.                         label.setText("Your " + textLabel + " is okay.");
792.                         label.setTextFill(Color.rgb(21, 117, 84));
793.                         return true;
794.                     }
795.                     return false;
796.                 }
797.                 public boolean checkUserForm(String username, String password, Strin
    g confirmPassword, String firstname, String lastname, String email, String userType
    ){
798.                     boolean checkUsername = checkEmptyForm(username, msgUsername, "u
    sername");
```

```
799.
800.                if(checkUsername && !editUserMode){
801.                        checkUsername = checkExistUsername(username, msgUsername, "u
    sername");
802.                    }
803.                    if(checkUsername && editUserMode){
804.                        if(!username.equals(selectedUserEdit.getUsername())){ // if
    username same
805.                            checkUsername = checkExistUsername(username, msgUsername
    , "username");
806.                        }
807.                    }
808.
809.                    boolean checkPassword = checkEmptyForm(password, msgPassword, "p
    assword");
810.                    boolean checkConfirmPassword = checkEmptyForm(confirmPassword, m
    sgConfirmPassword, "confirm password");
811.                    if(checkPassword && checkConfirmPassword){
812.                        checkConfirmPassword = checkPasswordMatch(password, confirmP
    assword, msgConfirmPassword, "confirm password");
813.                    }
814.
815.                    boolean checkFirstname = checkEmptyForm(firstname, msgFirstname,
     "firstname");
816.                    boolean checkLastname = checkEmptyForm(lastname, msgLastname, "l
    astname");
817.
818.                    boolean checkEmail = checkEmptyForm(email, msgEmail, "email");
819.
820.                    if(checkEmail){
821.                        checkEmail = isValidEmail(email, msgEmail, "email");
822.                        // in add mode
823.                        if(checkEmail && !editUserMode){
824.                           checkEmail = checkExistEmail(email, msgEmail, "email");
825.                        }
826.                        // in edit mode
827.                        if(checkEmail && editUserMode){
828.                            if(!email.equals(selectedUserEdit.getEmail())){ // if us
    ername same
829.                            checkEmail = checkExistEmail(email, msgEmail, "email");

830.                            }
831.                        }
832.                    }
833.
834.                    if(cbUserType.getValue() != null){ userType = cbUserType.getValu
    e(); }
835.
836.                boolean checkUserType = checkEmptyForm(userType, msgUserType, "user
    type");
837.
838.                    return checkUsername && checkPassword && checkConfirmPassword &&
     checkFirstname && checkLastname && checkEmail && checkUserType;
839.                }
840.
841.                public boolean checkUserEditForm(String username, String firstname,
    String lastname, String email, String userType){
842.                    boolean checkUsername = checkEmptyForm(username, msgUsername, "u
    sername");
843.                    if(checkUsername && !editUserMode){
844.                        checkUsername = checkExistUsername(username, msgUsername, "u
    sername");
845.                    }
846.                    if(checkUsername && editUserMode){
847.                        if(!username.equals(selectedUserEdit.getUsername())){ // if
    username same
```

```
848.                             checkUsername = checkExistUsername(username, msgUsername
    , "username");
849.                         }
850.                     }
851.
852.                     boolean checkFirstname = checkEmptyForm(firstname, msgFirstname,
     "firstname");
853.                     boolean checkLastname = checkEmptyForm(lastname, msgLastname, "l
    astname");
854.
855.                     boolean checkEmail = checkEmptyForm(email, msgEmail, "email");
856.
857.                     if(checkEmail){
858.                         checkEmail = isValidEmail(email, msgEmail, "email");
859.                         // in add mode
860.                         if(checkEmail && !editUserMode){
861.                            checkEmail = checkExistEmail(email, msgEmail, "email");
862.                         }
863.                         // in edit mode
864.                         if(checkEmail && editUserMode){
865.                             if(!email.equals(selectedUserEdit.getEmail())){ // if us
    ername same
866.                             checkEmail = checkExistEmail(email, msgEmail, "email");

867.                             }
868.                         }
869.                     }
870.
871.
872.                 if(cbUserType.getValue() != null){ userType = cbUserType.getValue();
     }
873.                     boolean checkUserType = checkEmptyForm(userType, msgUserType, "u
    ser type");
874.
875.                     return checkUsername && checkFirstname && checkLastname && check
    Email && checkUserType;
876.                 }
877.
878.                 public boolean checkUserPasswordForm(String password, String confirm
    Password){
879.                     boolean checkPassword = checkEmptyForm(password, msgPassword, "p
    assword");
880.                     boolean checkConfirmPassword = checkEmptyForm(confirmPassword, m
    sgConfirmPassword, "confirm password");
881.                     if(checkPassword && checkConfirmPassword){
882.                         checkConfirmPassword = checkPasswordMatch(password, confirmP
    assword, msgConfirmPassword, "confirm password");
883.                     }
884.
885.                     return checkPassword && checkConfirmPassword;
886.                 }
887.
888.                 public boolean checkUserSameValue(User editUser){
889.                     boolean checkUsername = editUser.getUsername().equals(selectedUs
    erEdit.getUsername());
890.                     boolean checkPassword = true; // if text == it equal
891.                     if(!txtPassword.getText().equals(selectedUserEdit.getPassword())
    ){ // if text not same in box
892.             //            System.out.println("Not same password");
893.                         checkPassword = editUser.getPassword().equals(selectedUserEd
    it.getPassword()); // hash and check is it same?
894.                     }
895.             //        else{
896.             //            System.out.println("Same password");
897.             //            checkPassword = true;
898.             //        }
```

```
899.                    boolean checkFirstname = editUser.getFirstname().equals(selected
    UserEdit.getFirstname());
900.                    boolean checkLastname = editUser.getLastname().equals(selectedUs
    erEdit.getLastname());
901.                    boolean checkEmail = editUser.getEmail().equals(selectedUserEdit
    .getEmail());
902.                    boolean checkUserType = editUser.getType().equals(selectedUserEd
    it.getType());
903.
904.
905.            //          System.out.println("Username : " + checkUsername);
906.            //          System.out.println("Password : " + checkPassword);
907.            //          System.out.println("Firstname : " + checkFirstname);
908.            //          System.out.println("Lastname : " + checkLastname);
909.            //          System.out.println("Email : " + checkEmail);
910.            //          System.out.println("UserType : " + checkUserType);
911.                    return checkUsername && checkPassword && checkFirstname && check
    Lastname && checkEmail && checkUserType;
912.                }
913.                public boolean checkUserSearchForm(String search){
914.                    boolean checkSearch = checkEmptyForm(search, msgUserSearch, "sea
    rch");
915.                    return checkSearch;
916.                }
917.                public void clearMovieSearchMessage(){
918.                    msgMovieSearch.setText(null);
919.                }
920.
921.                // Movie ========================================================
    ============================
922.                public void initMovieCol(){
923.                    colMovieId.setCellValueFactory(new PropertyValueFactory<>("id"))
    ;
924.                    colEnglishName.setCellValueFactory(new PropertyValueFactory<>("e
    nglishName"));
925.                    colThaiName.setCellValueFactory(new PropertyValueFactory<>("thai
    Name"));
926.                    colDirector.setCellValueFactory(new PropertyValueFactory<>("dire
    ctor"));
927.                    colCast.setCellValueFactory(new PropertyValueFactory<>("cast"));

928.                    colSynopsis.setCellValueFactory(new PropertyValueFactory<>("syno
    psis"));
929.                    colGenre.setCellValueFactory(new PropertyValueFactory<>("genre")
    );
930.                    colTime.setCellValueFactory(new PropertyValueFactory<>("time"));

931.                    colReleaseDate.setCellValueFactory(new PropertyValueFactory<>("r
    eleaseDate"));
932.                    colPoster.setCellValueFactory(new PropertyValueFactory<>("poster
    "));
933.                    colTrailer.setCellValueFactory(new PropertyValueFactory<>("trail
    er"));
934.                }
935.                public void loadMovieData(){
936.                    movieList.clear();
937.                    movieListArray = cc.getMovieList();
938.                    try {
939.
940.                    for (Movie movie : movieListArray) {
941.                            movieList.add(movie);
942.                        }
943.                    } catch (Exception e) {
944.                        System.out.println(e);
945.                    }
946.
```

```
947.                     tbMovie.setItems(movieList);
948.                     loadShowtimeSearchMovieCombo();
949.                 }
950.                 public void clearMovieMessage(){
951.                     msgEnglishName.setText(null);
952.                     msgThaiName.setText(null);
953.                     msgDirector.setText(null);
954.                     msgCast.setText(null);
955.                     msgSynopsis.setText(null);
956.                     msgGenre.setText(null);
957.                     msgHour.setText(null);
958.                     msgMinute.setText(null);
959.                     msgReleaseDate.setText(null);
960.                     msgPoster.setText(null);
961.                     msgTrailer.setText(null);
962.                 }
963.                 public void loadMovieSearchCombo(){
964.                     ObservableList<String> movieSearchOptions = FXCollections.observ
    ableArrayList("English Name","Thai Name","Director","Cast","Synopsis","Genre","Time
    ","Release Date");
965.                     cbMovieSearch.setItems(movieSearchOptions);
966.                     cbMovieSearch.setValue("English Name");
967.                 }
968.                 public void loadMovieGenreCombo(){
969.                     ObservableList<String> movieGenreOptions = FXCollections.observa
    bleArrayList("Action","Adventure","Comedy","Crime & Gangster","Drama","Epics/Histor
    ical","Horror",
970.                             "Musicals/Dance","Science Fiction","Romantic","War","Wes
    terns");
971.                     cbGenre.setItems(movieGenreOptions);
972.                 }
973.                 public void loadMovieHourCombo(){
974.                     ArrayList<String> hour = new ArrayList<String>();
975.                     for (int i = 0; i <= 10; i++) {
976.                         if(i<10){
977.                             hour.add("0"+String.valueOf(i));
978.                         }else{
979.                             hour.add(String.valueOf(i));
980.                         }
981.
982.                 }
983.                     ObservableList<String> movieHourOptions = FXCollections.observab
    leArrayList(hour);
984.                     cbTimeHour.setItems(movieHourOptions);
985.                 }
986.                 public void loadMovieMinuteCombo(){
987.                     ArrayList<String> minute = new ArrayList<String>();
988.                     for (int i = 0; i <= 59; i++) {
989.                         if(i<10){
990.                             minute.add("0"+String.valueOf(i));
991.                         }else{
992.                             minute.add(String.valueOf(i));
993.                         }
994.                     }
995.                     ObservableList<String> movieMinuteOptions = FXCollections.observ
    ableArrayList(minute);
996.                     cbTimeMinute.setItems(movieMinuteOptions);
997.                 }
998.                 public boolean checkMovieSearchForm(String search){
999.                     boolean checkSearch = checkEmptyForm(search, msgMovieSearch, "se
    arch");
1000.                    return checkSearch;
1001.                }
1002.                public String convertDate(DatePicker datePicker){
1003.                    String formattedString = null;
1004.                    try{
```

```
1005.                   LocalDate ld = datePicker.getValue();
1006.                   DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMM
    M yyyy");
1007.                   formattedString = ld.format(formatter);
1008.                   //labelPickerReleaseDate.setText(formattedString);
1009.                   }catch(Exception e){
1010.                       System.out.println(e);
1011.                   }
1012.                   return formattedString;
1013.               }
1014.               public boolean checkMovieForm(String englishName, String thaiName, S
    tring director, String cast, String synopsis, String genre, String movieHour ,Strin
    g movieMinute, String releaseDate, String poster, String trailer){
1015.                   boolean checkEnglishName = checkEmptyForm(englishName, msgEnglis
    hName, "english name");
1016.                   boolean checkThaiName = checkEmptyForm(thaiName, msgThaiName, "t
    hai name");
1017.                   boolean checkDirector = checkEmptyForm(director, msgDirector, "d
    irector");
1018.
1019.               boolean checkCast = checkEmptyForm(cast, msgCast, "cast");
1020.                   boolean checkSynopsis = checkEmptyForm(synopsis, msgSynopsis, "s
    ynopsis");
1021.
1022.                   if(cbGenre.getValue() != null){ genre = cbGenre.getValue(); }
1023.                   boolean checkGenre = checkEmptyForm(genre, msgGenre, "genre");
1024.
1025.                   // check hour and minute
1026.                   if(cbTimeHour.getValue() != null){ movieHour = cbTimeHour.getVal
    ue(); }
1027.                   boolean checkHour = checkEmptyForm(movieHour, msgHour, "hour");

1028.
1029.                   if(cbTimeMinute.getValue() != null){ movieMinute = cbTimeMinute.
    getValue(); }
1030.                   boolean checkMinute = checkEmptyForm(movieMinute, msgMinute, "mi
    nute");
1031.
1032.                   if(datePickerReleaseDate.getValue() != null){ releaseDate = conv
    ertDate(datePickerReleaseDate); }
1033.                   boolean checkReleaseDate = checkEmptyForm(releaseDate, msgReleas
    eDate, "release date");
1034.
1035.                   boolean checkPoster = false;
1036.                   if(selectedPoster != null){ poster = selectedPoster.toURI().toSt
    ring(); }
1037.                   if(!editMovieMode){
1038.                       checkPoster = checkEmptyForm(poster, msgPoster, "poster");
1039.                   }else{
1040.                       checkPoster = true;
1041.                   }
1042.
1043.                   boolean checkTrailer = checkEmptyForm(trailer, msgTrailer, "trai
    ler");
1044.
1045.                   return checkEnglishName && checkThaiName && checkDirector && che
    ckCast && checkSynopsis && checkGenre && checkHour && checkMinute && checkReleaseDa
    te && checkPoster && checkTrailer;
1046.               }
1047.               public void clearMovieForm(){
1048.                   txtEnglishName.clear();
1049.                   txtThaiName.clear();
1050.                   txtDirector.clear();
1051.                   txtCast.clear();
1052.                   txtSynopsis.clear();
1053.                   cbGenre.setValue(null);
```

```
1054.                   cbTimeHour.setValue(null);
1055.
1056.               cbTimeMinute.setValue(null);
1057.                   datePickerReleaseDate.setValue(null);
1058.                   selectedPoster = null; // unset image select
1059.                   imagePoster.setImage(null);
1060.                   txtTrailer.clear();
1061.               }
1062.
1063.               Movie selectedMovieEdit; // global variable
1064.               boolean editMovieMode = false;
1065.               int editMovieId = 0;
1066.
1067.               public void initMovieEditMode() throws ParseException{
1068.                   selectedMovieEdit = tbMovie.getSelectionModel().getSelectedItem(
    ); // เก็บมาเป็น object จาก list ที่เลือก
1069.                   if(selectedMovieEdit == null){
1070.                       AlertMaker.showErrorMessage("No movie selected", "Please sel
    ect a movie for edit.");
1071.                       return;
1072.                   }
1073.                   // Disable button to add & Enable button to save & Change head t
    ext to edit mode
1074.                   btnAddMovie.setDisable(true);
1075.                   btnSaveMovie.setDisable(false);
1076.                   btnCancleEditMovie.setDisable(false);
1077.
1078.                   lbHeadMovie.setText("Edit movie id : " + selectedMovieEdit.getId
    ());
1079.
1080.                   txtEnglishName.setText(selectedMovieEdit.getEnglishName());
1081.                   txtThaiName.setText(selectedMovieEdit.getThaiName());
1082.                   txtDirector.setText(selectedMovieEdit.getDirector());
1083.                   txtCast.setText(selectedMovieEdit.getCast());
1084.                   txtSynopsis.setText(selectedMovieEdit.getSynopsis());
1085.                   cbGenre.setValue(selectedMovieEdit.getGenre());
1086.                   // split time with colon
1087.                   String time = selectedMovieEdit.getTime();
1088.                   String hourminute[] = time.split(":");
1089.                   String hour = hourminute[0];
1090.                   String minute = hourminute[1];
1091.                   cbTimeHour.setValue(hour);
1092.                   cbTimeMinute.setValue(minute);
1093.
1094.                   DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMM
    M yyyy");
1095.                   String date = selectedMovieEdit.getReleaseDate();
1096.                   LocalDate localDate = LocalDate.parse(date, formatter);
1097.
1098.               System.out.println(localDate);
1099.                   datePickerReleaseDate.setValue(localDate);
1100.           //        datePickerReleaseDate.setValue(localDate);
1101.
1102.                   // poster image
1103.                   InputStream image = getClass().getResourceAsStream("/cinema/post
    erImage/"+selectedMovieEdit.getPoster());
1104.                   File imageFile = new File("/cinema/posterImage/"+selectedMovieEd
    it.getPoster());
1105.
1106.                   posterImage = new Image(image);
1107.                   imagePoster.setImage(posterImage);
1108.
1109.                   // Selected file from old path
1110.                   selectedPoster = imageFile;
1111.
1112.
```

```
1113.                   txtTrailer.setText(selectedMovieEdit.getTrailer());
1114.
1115.                   clearMovieMessage();
1116.
1117.                   msgPoster.setText(imageFile.getName());
1118.                   msgPoster.setTextFill(Color.BLACK);
1119.
1120.                   editMovieMode = true;
1121.                   editMovieId = tbMovie.getSelectionModel().getSelectedIndex();
1122.               }
1123.               public void cancleMovieEdit(){
1124.                   btnAddMovie.setDisable(false);
1125.                   btnSaveMovie.setDisable(true);
1126.                   btnCancleEditMovie.setDisable(true);
1127.                   lbHeadMovie.setText("Add new movie");
1128.                   clearMovieForm();
1129.                   clearMovieMessage();
1130.                   editMovieMode = false;
1131.               }
1132.               public boolean checkMovieSameValue(Movie editMovie){
1133.                   boolean checkEnglishName = editMovie.getEnglishName().equals(sel
    ectedMovieEdit.getEnglishName());
1134.                   boolean checkThaiName = editMovie.getThaiName().equals(selectedM
    ovieEdit.getThaiName());
1135.                   boolean checkDirector = editMovie.getDirector().equals(selectedM
    ovieEdit.getDirector());
1136.                   boolean checkCast = editMovie.getCast().equals(selectedMovieEdit
    .getCast());
1137.
1138.               boolean checkSynopsis = editMovie.getSynopsis().equals(selectedMovie
    Edit.getSynopsis());
1139.                   boolean checkGenre = editMovie.getGenre().equals(selectedMovieEd
    it.getGenre());
1140.                   boolean checkTime = editMovie.getTime().equals(selectedMovieEdit
    .getTime());
1141.
1142.                   boolean checkReleaseDate = editMovie.getReleaseDate().equals(sel
    ectedMovieEdit.getReleaseDate());
1143.                   System.out.println("MSG "+msgPoster.getText());
1144.                   System.out.println("EDIT "+selectedMovieEdit.getPoster());
1145.                   boolean checkPoster = msgPoster.getText().equals(selectedMovieEd
    it.getPoster());
1146.                   boolean checkTrailer = editMovie.getTrailer().equals(selectedMov
    ieEdit.getTrailer());
1147.
1148.                   System.out.println(String.valueOf(checkEnglishName));
1149.                   System.out.println(String.valueOf(checkThaiName));
1150.                   System.out.println(String.valueOf(checkDirector));
1151.                   System.out.println(String.valueOf(checkCast));
1152.                   System.out.println(String.valueOf(checkSynopsis));
1153.                   System.out.println(String.valueOf(checkGenre));
1154.                   System.out.println(String.valueOf(checkTime));
1155.                   System.out.println(String.valueOf(checkReleaseDate));
1156.                   System.out.println(String.valueOf(checkPoster));
1157.                   System.out.println(String.valueOf(checkTrailer));
1158.
1159.
1160.                   return checkEnglishName && checkThaiName && checkDirector && che
    ckCast && checkSynopsis && checkGenre && checkTime && checkReleaseDate && checkPost
    er && checkTrailer;
1161.               }
1162.
1163.               // Promotion ----------------------------------------------------
    -------
1164.               public void initPromotionCol(){
```

```
1165.                   colPromotionId.setCellValueFactory(new PropertyValueFactory<>("p
    romotionID"));
1166.                   colPromotionName.setCellValueFactory(new PropertyValueFactory<>(
    "name"));
1167.                   colDate.setCellValueFactory(new PropertyValueFactory<>("remainin
    gDate"));
1168.                   colDescription.setCellValueFactory(new PropertyValueFactory<>("d
    escription"));
1169.                   colDiscount.setCellValueFactory(new PropertyValueFactory<>("disc
    ount"));
1170.
1171.
1172.           }
1173.               public void loadPromotionData() {
1174.                   promotionList.clear();
1175.                   promotionListArray = pc.getPromotionList();
1176.                   try {
1177.                       for (Promotion p : promotionListArray) {
1178.                           promotionList.add(p);
1179.
1180.                       }
1181.                   } catch (Exception e) {
1182.                       System.out.println(e);
1183.                   }
1184.                   tbPromotion.setItems(promotionList);
1185.               }
1186.               public void clearPromotionMessage() {
1187.                   msgPromotion.setText(null);
1188.                   msgDescription.setText(null);
1189.                   msgDate.setText(null);
1190.                   msgDiscount.setText(null);
1191.               }
1192.               public void clearPromotionForm() {
1193.                   txtPromotion.clear();
1194.                   txtDescription.clear();
1195.                   pickerDate.setValue(null);
1196.                   txtDiscount.clear();
1197.               }
1198.               public boolean isNumeric(String str, Label label, String textLabel){

1199.                   if (!str.matches("-?\\d+(\\.\\d+)?")) {
1200.                       label.setText("Your " + textLabel + " is not number!");
1201.                       label.setTextFill(Color.rgb(210, 39, 30));
1202.                   } else {
1203.                       label.setText("Your " + textLabel + " is okay.");
1204.                       label.setTextFill(Color.rgb(21, 117, 84));
1205.                       return true;
1206.                   }
1207.                   return false;
1208.               }
1209.               public boolean checkPromotionForm(String name, String description, S
    tring date, String discount) {
1210.                   boolean checkName = checkEmptyForm(name, msgPromotion, "promotio
    n name");
1211.                   boolean checkDesc = checkEmptyForm(description, msgDescription,
    "description");
1212.                   boolean checkDisc = checkEmptyForm(discount, msgDiscount, "disco
    unt");
1213.                   if(checkDisc){
1214.
1215.                   checkDisc = isNumeric(discount, msgDiscount, "discount");
1216.                   }
1217.                   if (date == null) {date = "";}
1218.                   boolean checkDate = checkEmptyForm(date, msgDate, "remaining dat
    e");
1219.                   return checkName && checkDesc && checkDate && checkDisc;
```

```
1220.               }
1221.
1222.               Promotion selectedPromotionEdit;
1223.               boolean editPromotionMode = false;
1224.               int editPromotionID = 0;
1225.
1226.               public void initPromotionEdit() throws ParseException {
1227.                   selectedPromotionEdit = tbPromotion.getSelectionModel().getSelec
    tedItem();
1228.                   if(selectedPromotionEdit == null) {
1229.                       AlertMaker.showErrorMessage("No promotion selected", "Please
     select a promotion to edit");
1230.                       return;
1231.                   }
1232.
1233.                   // Disable add button
1234.                   btnAddPromotion.setDisable(true);
1235.                   btnSavePromotion.setDisable(false);
1236.                   btnCancelPromotion.setDisable(false);
1237.
1238.                   lbHeadPromotion.setText("Edit promotion id : " + selectedPromoti
    onEdit.getPromotionID());
1239.
1240.                   txtPromotion.setText(selectedPromotionEdit.getName());
1241.                   txtDescription.setText(selectedPromotionEdit.getDescription());

1242.                   txtDiscount.setText(selectedPromotionEdit.getDiscount()+"");
1243.
1244.
1245.                   DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMM
    M yyyy");
1246.                   LocalDate localdate = LocalDate.parse(selectedPromotionEdit.getR
    emainingDate(), formatter);
1247.                   System.out.println(localdate);
1248.                   pickerDate.setValue(localdate);
1249.
1250.                   clearPromotionMessage();
1251.                   editPromotionMode = true;
1252.                   editPromotionID = tbPromotion.getSelectionModel().getSelectedInd
    ex();
1253.               }
1254.
1255.           public void cancelPromotionEdit() {
1256.                   btnAddPromotion.setDisable(false);
1257.                   btnSavePromotion.setDisable(true);
1258.                   btnCancelPromotion.setDisable(true);
1259.
1260.                   lbHeadPromotion.setText("Add new promotion");
1261.                   clearPromotionForm();
1262.                   editPromotionMode = false;
1263.               }
1264.               public boolean checkPromotionSearchForm(String s) {
1265.                   boolean checkSearch = checkEmptyForm(s, msgPromotionSearch, "sea
    rch");
1266.                   return checkSearch;
1267.               }
1268.               public void clearPromotionSearchMessage(){
1269.                   msgPromotionSearch.setText(null);
1270.               }
1271.
1272.               /*
1273.               initTheatreCol();
1274.               loadTheatreData();
1275.
1276.               */
```

```
1277.               // Theatre ------------------------------------------------------------
    -----
1278.               public boolean checkInt(String str, Label label, String textLabel){

1279.                   if(!str.matches("[0-9]+")){
1280.                       label.setText("Your " + textLabel + " is not integer!");
1281.                       label.setTextFill(Color.rgb(210, 39, 30));
1282.                   }else{
1283.                       label.setText("Your " + textLabel + " is okay.");
1284.                       label.setTextFill(Color.rgb(21, 117, 84));
1285.                       return true;
1286.                   }
1287.                   return false;
1288.               }
1289.               public void initTheatreCol(){
1290.                   colTheatreId.setCellValueFactory(new PropertyValueFactory<>("id"
    ));
1291.                   colTheatreNumber.setCellValueFactory(new PropertyValueFactory<>(
    "theatreNumber"));
1292.                   colTheatreType.setCellValueFactory(new PropertyValueFactory<>("t
    heatreType"));
1293.               }
1294.               public void loadTheatreData() {
1295.                   theatreList.clear();
1296.
1297.               theatreListArray = cc.getTheatreList();
1298.                   try {
1299.                       for (Theatre t : theatreListArray) {
1300.                           theatreList.add(t);
1301.                       }
1302.                   } catch (Exception e) {
1303.                       System.out.println(e);
1304.                   }
1305.                   tbTheatre.setItems(theatreList);
1306.                   loadTheatreShowtimeCombo();
1307.               }
1308.               public void clearTheatreMessage() {
1309.                   msgTheatre.setText(null);
1310.               }
1311.               public void clearTheatreForm() {
1312.                   txtTheatre.clear();
1313.                   cbTheatreType.setValue("2D/3D");
1314.               }
1315.
1316.               public void loadTheatreTypeCombo(){
1317.                   ObservableList<String> theatreTypeOptions = FXCollections.observ
    ableArrayList("2D/3D","4D");
1318.                   cbTheatreType.setItems(theatreTypeOptions);
1319.                   cbTheatreType.setValue("2D/3D");
1320.               }
1321.
1322.               public boolean checkTheatreForm(String number) {
1323.                   boolean checkTheatre = checkEmptyForm(number, msgTheatre, "theat
    re number");
1324.                   if(checkTheatre){
1325.                       checkTheatre = checkInt(number, msgTheatre, "theatre number"
    );
1326.                   }
1327.                   return checkTheatre;
1328.               }
1329.
1330.               Theatre selectedTheatreEdit;
1331.               boolean editTheatreMode = false;
1332.               int editTheatreID = 0;
1333.
1334.               public void initTheatreEdit() {
```

```
1335.                   selectedTheatreEdit = tbTheatre.getSelectionModel().getSelectedI
    tem();
1336.                   if(selectedTheatreEdit == null) {
1337.                       AlertMaker.showErrorMessage("No theatre selected", "Please s
    elect a theatre to edit");
1338.                       return;
1339.
1340.               }
1341.
1342.                   // Disable add button
1343.                   btnAddTheatre.setDisable(true);
1344.                   btnSaveTheatre.setDisable(false);
1345.                   btnCancelTheatre.setDisable(false);
1346.
1347.                   lbHeadTheatre.setText("Edit theatre id : " + selectedTheatreEdit
    .getId());
1348.
1349.                   txtTheatre.setText(String.valueOf(selectedTheatreEdit.getTheatre
    Number()));
1350.                   cbTheatreType.setValue(selectedTheatreEdit.getTheatreType());
1351.
1352.                   clearTheatreMessage();
1353.                   editTheatreMode = true;
1354.                   editTheatreID = tbTheatre.getSelectionModel().getSelectedIndex()
    ;
1355.               }
1356.               public void cancelTheatreEdit() {
1357.                   btnAddTheatre.setDisable(false);
1358.                   btnSaveTheatre.setDisable(true);
1359.                   btnCancelTheatre.setDisable(true);
1360.
1361.                   lbHeadTheatre.setText("Add new theatre");
1362.                   clearTheatreForm();
1363.                   editTheatreMode = false;
1364.               }
1365.
1366.
1367.               // Showtime=============================================================
    ==============
1368.                // Showtime=============================================================
    =============
1369.               public void initShowtimeCol(){
1370.                   colShowtimeId1.setCellValueFactory(new PropertyValueFactory<>("i
    d"));
1371.                   colMoviesName1.setCellValueFactory(new PropertyValueFactory<>("m
    ovie"));
1372.                   colTheatreShow1.setCellValueFactory(new PropertyValueFactory<>("
    theatre"));
1373.                   colIncreaseSeatPrice1.setCellValueFactory(new PropertyValueFacto
    ry<>("increaseSeatPrice"));
1374.
1375.               colSoundtrackShow1.setCellValueFactory(new PropertyValueFactory<>("s
    oundtrack"));
1376.                   colShowtimeSystem.setCellValueFactory(new PropertyValueFactory<>
    ("system"));
1377.                   colPeriodShow1.setCellValueFactory(new PropertyValueFactory<>("s
    tartTime"));
1378.                   colShowtimeDate.setCellValueFactory(new PropertyValueFactory<>("
    date"));
1379.                   colSubtitle.setCellValueFactory(new PropertyValueFactory<>("subt
    itle"));
1380.               }
1381.               public void loadShowtimeData(){
1382.                   showtimeList.clear();
1383.                   showtimeListArray = cc.getShowtimeList();
1384.                   try {
```

```
1385.                       for (Showtime showtime : showtimeListArray) {
1386.                           showtimeList.add(showtime);
1387.                       }
1388.                   } catch (Exception e) {
1389.                       System.out.println(e);
1390.                   }
1391.
1392.                   tbShowtime1.setItems(showtimeList);
1393.               }
1394.               public void clearShowtimeMessage(){
1395.                   msgIncreaseSeatPrice1.setText(null);
1396.                   msgMovieSearch1.setText(null);
1397.                   msgTheatreShow1.setText(null);
1398.                   msgTimeShowtime.setText(null);
1399.                   msgShowtimeDate.setText(null);
1400.               }
1401.
1402.               public void loadShowtimeSearchMovieCombo(){
1403.                   ObservableList<String> movieShowtimeList = FXCollections.observa
    bleArrayList();
1404.                   movieListArray = cc.getMovieList();
1405.                   try {
1406.                       for (Movie movie : movieListArray) {
1407.                           movieShowtimeList.add(movie.getId() + " : " + movie.getE
    nglishName());
1408.                       }
1409.                   } catch (Exception e) {
1410.                       System.out.println(e);
1411.                   }
1412.                   cbShowtimeSearch1.setItems(movieShowtimeList);
1413.               }
1414.
1415.
1416.
1417.               public void loadSoundtrackCombo(){
1418.                   ObservableList<String> soundTrack = FXCollections.observableArra
    yList("EN","TH");
1419.                   cbSoundtrack1.setItems(soundTrack);
1420.                   cbSoundtrack1.setValue("TH");
1421.               }
1422.
1423.               public void loadStartHourCombo(){
1424.                   ArrayList<String> hour = new ArrayList<String>();
1425.                   for (int i = 0; i <= 23; i++) {
1426.                       if(i<10){
1427.                           hour.add("0"+String.valueOf(i));
1428.                       }else{
1429.                           hour.add(String.valueOf(i));
1430.                       }
1431.                   }
1432.                   ObservableList<String> showHourOptions = FXCollections.observabl
    eArrayList(hour);
1433.                   cbTimeHourShow1.setItems(showHourOptions);
1434.           //        cbTimeHourShow1.setValue("None");
1435.               }
1436.               public void loadStartMinCombo(){
1437.                   ArrayList<String> min = new ArrayList<String>();
1438.                   for (int i = 0; i <= 55; i++) {
1439.                       if(i % 5 == 0){
1440.                           if(i<10){
1441.                               min.add("0"+String.valueOf(i));
1442.                           }else{
1443.                               min.add(String.valueOf(i));
1444.                           }
1445.                       }
1446.                   }
```

```
1447.                   ObservableList<String> showMinOptions = FXCollections.observable
    ArrayList(min);
1448.                   cbTimeMinuteShow1.setItems(showMinOptions);
1449.                   cbTimeMinuteShow1.setValue("");
1450.               }
1451.
1452.               public void loadTheatreShowtimeCombo(){
1453.                   ObservableList<String> theatreNum = FXCollections.observableArra
    yList();
1454.                   List<Theatre> theatre = new ArrayList<Theatre>();
1455.                   theatre = cc.getTheatreList();
1456.
1457.
1458.               try {
1459.                       for (Theatre t : theatre) {
1460.                           theatreNum.add(t.getId()+ " : Theatre " + t.getTheatreNu
    mber());
1461.                       }
1462.                   } catch (Exception e) {
1463.                       System.out.println(e);
1464.                   }
1465.
1466.                   cbTheatreSearch1.setItems(theatreNum);
1467.                   //cbTheatreSearch1.setValue("1");
1468.               }
1469.
1470.               // Check theatre and load combo
1471.               boolean changeTheatre = false; // t คือ มีการเปลี่ยน f คือไม่มีการเปลี่ยน
1472.               public void loadSystemCombo(){
1473.                   ObservableList<String> system = FXCollections.observableArrayLis
    t();
1474.                   Theatre t;
1475.                   // Find theatre
1476.                   if(!editShowtime){
1477.                       String[] selectTheId = cbTheatreSearch1.getValue().split(" :
    ");
1478.                       //System.out.println("Theatre ID: " + selectTheId[0]);
1479.                       selectTheId[0].replaceAll("\\s+","");
1480.                       int theatre_id = Integer.valueOf(selectTheId[0]);
1481.                       t = cc.getTheatre(Integer.valueOf(theatre_id));
1482.                       selectShowtimeSystem(system,t);
1483.                   }else{
1484.                       if(!changeTheatre){
1485.                           t = selectedShowtimeEdit.getTheatre();
1486.                           selectShowtimeSystem(system,t);
1487.                       }else{
1488.                           String[] selectTheId = cbTheatreSearch1.getValue().split
    (" :");
1489.                           //System.out.println("Theatre ID: " + selectTheId[0]);
1490.                           selectTheId[0].replaceAll("\\s+","");
1491.                           int theatre_id = Integer.valueOf(selectTheId[0]);
1492.                           t = cc.getTheatre(Integer.valueOf(theatre_id));
1493.                           selectShowtimeSystem(system,t);
1494.                       }
1495.                   }
1496.
1497.                   // No Select
1498.
1499.
1500.               }
1501.
1502.
1503.               public void selectShowtimeSystem(ObservableList<String> system, Thea
    tre t){
1504.                   if(t.getTheatreType().equals("2D/3D")){
1505.                       system = FXCollections.observableArrayList("2D","3D");
```

```
1506.                       cbSystem.setItems(system);
1507.                       cbSystem.setValue("2D");
1508.                   }else{
1509.                       system = FXCollections.observableArrayList("4DX");
1510.                       cbSystem.setItems(system);
1511.                       cbSystem.setValue("4DX");
1512.                   }
1513.               }
1514.
1515.               public void clearShowtimeForm(){
1516.                   txtIncreaseSeatPrice1.clear();
1517.                   cbShowtimeSearch1.setValue(null);
1518.                   cbTheatreSearch1.setValue(null);
1519.                   cbSoundtrack1.setValue("TH");
1520.                   cbTimeHourShow1.setValue(null);
1521.                   cbTimeMinuteShow1.setValue(null);
1522.                   //cbSystem.setValue("2D");
1523.                   datePickerShowtime.setValue(null);
1524.               }
1525.
1526.               Showtime selectedShowtimeEdit; // global variable
1527.               boolean editShowtime = false;
1528.               int editShowtimeId = 0;
1529.
1530.               public void initShowtimeEditMode() throws ParseException{
1531.
1532.
1533.                   selectedShowtimeEdit = tbShowtime1.getSelectionModel().getSelect
    edItem(); // เก็บมาเป็น object จาก list ที่เลือก
1534.                   if(selectedShowtimeEdit == null){
1535.                       AlertMaker.showErrorMessage("No showtime selected", "Please
    select a showtime for edit.");
1536.                       return;
1537.                   }
1538.
1539.
1540.                   // Disable button to add & Enable button to save & Change head t
    ext to edit mode
1541.                   btnAddShowtime1.setDisable(true);
1542.                   btnSaveShowtime1.setDisable(false);
1543.                   btnCancleEditShowtime1.setDisable(false);
1544.
1545.
1546.                   lbHeadShowtime1.setText("Edit showtime id : " + String.valueOf(s
    electedShowtimeEdit.getId()));
1547.
1548.                   txtIncreaseSeatPrice1.setText(String.valueOf(selectedShowtimeEdi
    t.getIncreaseSeatPrice()));
1549.                   String selectMov = selectedShowtimeEdit.getMovie().getId()  + "
    : " + selectedShowtimeEdit.getMovie().getEnglishName();
1550.                   cbShowtimeSearch1.setValue(selectMov);
1551.                   String[] selectMovId = selectMov.split(":");
1552.                   System.out.println("Movie ID: " + selectMovId[0]);
1553.
1554.                   String selectThe = selectedShowtimeEdit.getTheatre().getId()+ "
    : Theatre " + selectedShowtimeEdit.getTheatre().getTheatreNumber();
1555.                   cbTheatreSearch1.setValue(selectThe);
1556.                   String[] selectTheId = selectThe.split(":");
1557.                   System.out.println("Theatre ID: " + selectTheId[0]);
1558.
1559.                   cbSoundtrack1.setValue(selectedShowtimeEdit.getSoundtrack());
1560.                   // split time with colon
1561.                   String period = selectedShowtimeEdit.getStartTime();
1562.                   String houraminute[] = period.split(":");
1563.                   String hour = houraminute[0];
1564.
```

```
1565.                   String min = houraminute[1];
1566.                   String minUse = min.substring(0,2);
1567.                   System.out.println("Min"+minUse);
1568.                   cbTimeHourShow1.setValue(hour);
1569.                   cbTimeMinuteShow1.setValue(minUse);
1570.
1571.
1572.
1573.                   DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMM
    M yyyy");
1574.                   String date = selectedShowtimeEdit.getDate();
1575.                   LocalDate localDate = LocalDate.parse(date, formatter);
1576.                   System.out.println(localDate);
1577.                   datePickerShowtime.setValue(localDate);
1578.
1579.                   clearShowtimeMessage();
1580.
1581.                   // load new combo
1582.                   changeTheatre = false;
1583.                   editShowtime = true;
1584.                   loadSystemCombo();
1585.
1586.               cbSystem.setValue(selectedShowtimeEdit.getSystem());
1587.
1588.                   editShowtimeId = tbShowtime1.getSelectionModel().getSelectedInde
    x();
1589.               }
1590.               public void cancleShowtimeEdit(){
1591.                   btnAddShowtime1.setDisable(false);
1592.                   btnSaveShowtime1.setDisable(true);
1593.                   btnCancleEditShowtime1.setDisable(true);
1594.                   lbHeadShowtime1.setText("Add new Showtime");
1595.                   clearShowtimeForm();
1596.                   editShowtime = false;
1597.                   cbSystem.setDisable(true);
1598.                   //loadSystemCombo();
1599.               }
1600.               public boolean checkShowtimeForm(String movieName, String theatre, S
    tring increaseSP,String date,String min,String hour,String sound){
1601.                   boolean checkIncrease = checkEmptyForm(increaseSP, msgIncreaseSe
    atPrice1, "increase seat price");
1602.                   if(checkIncrease){
1603.                       checkIncrease = isNumeric(increaseSP, msgIncreaseSeatPrice1,
     "increase seat price");
1604.                       if(checkIncrease){
1605.                           double inc_double = Double.parseDouble(txtIncreaseSeatPr
    ice1.getText());
1606.                           if (inc_double < 0) {
1607.                               msgIncreaseSeatPrice1.setText("Your increase seat pr
    ice is less than 0!");
1608.                               msgIncreaseSeatPrice1.setTextFill(Color.rgb(210, 39,
     30));
1609.                               checkIncrease = false;
1610.                           }else{
1611.                               msgIncreaseSeatPrice1.setText("Your increase seat pr
    ice is okay.");
1612.                               msgIncreaseSeatPrice1.setTextFill(Color.rgb(21, 117,
     84));
1613.                               checkIncrease = true;
1614.                           }
1615.                       }
1616.                   }
1617.
1618.                   if(datePickerShowtime.getValue() != null){ date = convertDate(da
    tePickerShowtime); }
```

```
1619.                   boolean checkShowtimeDate = checkEmptyForm(date, msgShowtimeDate
    , "showtime date");
1620.
1621.
1622.
1623.
1624.                   if(cbShowtimeSearch1.getValue() != null){ movieName = cbShowtime
    Search1.getValue(); }
1625.                   boolean checkMoviename = checkEmptyForm(movieName, msgMovieSearc
    h1, "moive name");
1626.
1627.                   if(cbTheatreSearch1.getValue() != null){ theatre = cbTheatreSear
    ch1.getValue(); }
1628.                   boolean checktheatre = checkEmptyForm(theatre, msgTheatreShow1,
    "theatre");
1629.
1630.                   // check hour and minute
1631.                   if(cbTimeHourShow1.getValue() != null){ hour = cbTimeHourShow1.g
    etValue(); }
1632.                   if(cbTimeMinuteShow1.getValue() != null){ min = cbTimeMinuteShow
    1.getValue(); }
1633.
1634.                   boolean checkHour = checkEmptyForm(hour, msgTimeShowtime, "time"
    );
1635.                   boolean checkMinute = checkEmptyForm(min, msgTimeShowtime, "time
    ");
1636.                   boolean checkTime = checkHour && checkMinute;
1637.                   if (checkTime) {} else { checkTime = checkEmptyForm("", msgTimeS
    howtime, "time"); }
1638.
1639.
1640.
1641.                   if(cbSoundtrack1.getValue() != null){ sound = cbSoundtrack1.getV
    alue(); }
1642.
1643.                   return checkIncrease && checkMoviename &&  checktheatre && check
    ShowtimeDate && checkTime;
1644.               }
1645.
1646.               // Booking ================================================================
    ============
1647.               public void initBookingCol(){
1648.                   colBookingID.setCellValueFactory(new PropertyValueFactory<>("id"
    ));
1649.                   colBookingCreated.setCellValueFactory(new PropertyValueFactory<>
    ("BookingCreateDatetime"));
1650.                   colBookingUpdate.setCellValueFactory(new PropertyValueFactory<>(
    "BookingUpdateDatetime"));
1651.                   colBookingShowtime.setCellValueFactory(new PropertyValueFactory<
    >("ShowtimeDetail"));
1652.                   colBookingUser.setCellValueFactory(new PropertyValueFactory<>("U
    serDetail"));
1653.
1654.               colBookingPromotion.setCellValueFactory(new PropertyValueFactory<>("
    promotion"));
1655.                   colBookingStatus.setCellValueFactory(new PropertyValueFactory<>(
    "isCancel"));
1656.                   colBookingSeat.setCellValueFactory(new PropertyValueFactory<>("B
    ookedSeatString"));
1657.                   colBookingTotal.setCellValueFactory(new PropertyValueFactory<>("
    totalCost"));
1658.               }
1659.               public void loadBookingData(){
1660.                   bookingList.clear();
1661.                   bookingListArray = cc.getBookingList();
1662.                   try {
```

```
1663.                       for (Booking booking : bookingListArray) {
1664.                           bookingList.add(booking);
1665.                       }
1666.                   } catch (Exception e) {
1667.                       System.out.println(e);
1668.                   }
1669.
1670.                   tbBooking.setItems(bookingList);
1671.               }
1672.               public void setBookingTotal(){
1673.                   txtSummary.setText("Total Income : " + cc.getSumTotal() + " Baht
    ");
1674.               }
1675.
1676.               // User ========================================================
    ===========
1677.               @FXML
1678.               private void handleUserSearch(ActionEvent event) {
1679.                   String userSearch = txtUserSearch.getText();
1680.                   boolean isSearchOK = checkUserSearchForm(userSearch);
1681.                   if(isSearchOK){
1682.                       // System.out.println("Search Ready");
1683.                       String searchUserOf = cbUserSearch.getValue();
1684.                       // System.out.println(searchUserOf.toLowerCase());
1685.                       uc.searchUser(userSearch, searchUserOf.toLowerCase()); // se
    arch
1686.                       clearUserSearchMessage();
1687.                       // clear list
1688.                       userSearchList.clear();
1689.
1690.                       userSearchListArray = uc.getUserSearchList();
1691.                       try {
1692.                           for (User user : userSearchListArray) {
1693.
1694.                           userSearchList.add(user);
1695.                           }
1696.                       } catch (Exception e) {
1697.                           System.out.println(e);
1698.                       }
1699.                       if(userSearchList.size() > 0){ // Found
1700.                           tbUser.setItems(userSearchList);
1701.                           msgUserSearch.setText("Found!");
1702.                           msgUserSearch.setTextFill(Color.rgb(21, 117, 84));
1703.                           btnCancleUserSearch.setDisable(false);
1704.                           cancleUserEdit(); // after search cancle edit
1705.                       }else{
1706.                           msgUserSearch.setText("Not found!");
1707.                           msgUserSearch.setTextFill(Color.rgb(210, 39, 30));
1708.                           btnCancleUserSearch.setDisable(true);
1709.                           loadUserData();
1710.                       }
1711.                   }else{
1712.                       loadUserData();
1713.                   }
1714.               }
1715.               @FXML
1716.               private void handleCancleUserSearch(ActionEvent event) {
1717.                   // load normal table without search
1718.                   loadUserData();
1719.                   clearUserSearchMessage();
1720.                   txtUserSearch.setText(null);
1721.                   btnCancleUserSearch.setDisable(true);
1722.               }
1723.               @FXML
1724.               private void handleAddUser(ActionEvent event) {
1725.                   String username = txtUsername.getText();
```

```
1726.                   String password = txtPassword.getText();
1727.                   String confirmPassword = txtConfirmPassword.getText();
1728.                   String firstname = txtFirstname.getText();
1729.                   String lastname = txtLastname.getText();
1730.                   String email = txtEmail.getText();
1731.                   String userType = ""; // global
1732.
1733.                   boolean isUserOK = checkUserForm(username, password, confirmPass
    word, firstname, lastname, email, userType);
1734.                   System.out.println("Edit mode " + String.valueOf(editUserMode));

1735.                   if(isUserOK){
1736.                       userType = cbUserType.getValue(); // get value from combobox

1737.                       User newUser = new User(username, password, firstname, lastn
    ame, email, userType,0.0);
1738.
1739.                   newUser.setEncryptPassword();
1740.                       AlertMaker.showSimpleAlert("OK", newUser.toString());
1741.                       uc.addUser(newUser); // add new user
1742.                       clearUserMessage(); // clear label
1743.                       clearUserForm(); // clear text form
1744.                       loadUserData(); // load new list after add
1745.                   }
1746.               }
1747.               @FXML
1748.               private void handleSaveUser(ActionEvent event) {
1749.                   String username = txtUsername.getText();
1750.                   String firstname = txtFirstname.getText();
1751.                   String lastname = txtLastname.getText();
1752.                   String email = txtEmail.getText();
1753.                   String userType = ""; // global
1754.
1755.                   boolean isUserOK = checkUserEditForm(username, firstname, lastna
    me, email, userType);
1756.
1757.                   System.out.println("Edit user mode " + String.valueOf(editUserMo
    de));
1758.
1759.                   if(isUserOK){
1760.                       System.out.println("Edit mode start...");
1761.                       userType = cbUserType.getValue(); // get value from combobox

1762.
1763.                       User editUser = new User(username, null, firstname, lastname
    , email, userType,0.0);
1764.
1765.
1766.                       boolean isUserSame = checkUserSameValue(editUser);
1767.                       if(!isUserSame){ // have change
1768.                           uc.editUser(selectedUserEdit.getId(), editUser);
1769.                           System.out.println("Not same");
1770.                           loadUserData(); // load new list after add
1771.                           // update edit select to new information
1772.                           selectedUserEdit = tbUser.getItems().get(editUserId);
1773.                           AlertMaker.showSimpleAlert("Saved!", "Save information c
    ompleted");
1774.
1775.                       }else{
1776.                           AlertMaker.showSimpleAlert("No change!", "No changed for
     this user information");
1777.                       }
1778.
1779.                   }
1780.               }
1781.
```

```
1782.           @FXML
1783.               private void handleCancleEditUser(ActionEvent event) {
1784.                   cancleUserEdit();
1785.               }
1786.               @FXML
1787.               private void handleUserRefresh(ActionEvent event) {
1788.                   loadUserData();
1789.               }
1790.               @FXML
1791.               private void handleUserEdit(ActionEvent event) {
1792.                   initUserEditMode();
1793.                   System.out.println("Selected edit id : "+editUserId);
1794.               }
1795.               @FXML
1796.               private void handleUserDelete(ActionEvent event) {
1797.                   User selectedUserDelete = tbUser.getSelectionModel().getSelected
    Item(); // เก็บมาเป็น object จาก list ที่เลือก
1798.                   if(selectedUserDelete == null){
1799.                       AlertMaker.showErrorMessage("No user selected", "Please sele
    ct a user for delete.");
1800.                       return;
1801.                   }
1802.
1803.                   Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
1804.                   alert.setTitle("Deleting member");
1805.                   alert.setContentText("Are you sure want to delete the member "+
    selectedUserDelete.getUsername() + " ?");
1806.                   Optional<ButtonType> answer = alert.showAndWait();
1807.
1808.                   if(answer.get() == ButtonType.OK){
1809.                       uc.deleteUser(selectedUserDelete.getId());
1810.                       cancleUserEdit();
1811.                       loadUserData();
1812.                   }
1813.               }
1814.               @FXML
1815.               private void handleMouseClickUserEdit(MouseEvent event) {
1816.                   if (event.isPrimaryButtonDown() && event.getClickCount() == 2) {

1817.                       initUserEditMode();
1818.                       //System.out.println(selectedUserEdit);
1819.                       System.out.println("Selected edit user id : "+editUserId);
1820.                   }
1821.               }
1822.
1823.               boolean editUserPassword = false;
1824.               public void initChangeUserPassword(){
1825.
1826.               txtUsername.setDisable(true);
1827.                       txtPassword.setDisable(true);
1828.
1829.
1830.                       txtPassword.clear();
1831.                       txtConfirmPassword.clear();
1832.                       txtPassword.setDisable(false);
1833.                       txtConfirmPassword.setDisable(false);
1834.
1835.
1836.                       txtFirstname.setDisable(true);
1837.                       txtLastname.setDisable(true);
1838.                       txtEmail.setDisable(true);
1839.                       cbUserType.setDisable(true);
1840.                       editUserPassword = true;
1841.                       btnSaveUser.setDisable(true);
1842.
1843.               }
```

```
1844.               @FXML
1845.               private void handleChangePassword(ActionEvent event) {
1846.                   if(editUserPassword == false){
1847.                       initChangeUserPassword();
1848.                       System.out.println("Pre edit");
1849.                       editUserPassword = true;
1850.                   }else{
1851.
1852.                       String password = txtPassword.getText();
1853.                       String confirmPassword = txtConfirmPassword.getText();
1854.
1855.
1856.                       boolean isPassOK = checkUserPasswordForm(password, confirmPa
    ssword);
1857.                       if (isPassOK) {
1858.                           System.out.println("Ready for edit");
1859.                           //editUserPassword = false;
1860.                           initChangeUserPassword();
1861.                           clearUserMessage();
1862.                           uc.editUserPassword(selectedUserEdit.getId(), User.getEn
    cryptPassword(password));
1863.                           loadUserData(); // load new list after add
1864.                           AlertMaker.showSimpleAlert("Ok", "Password Ok");
1865.                       }
1866.
1867.                   }
1868.
1869.
1870.
1871.
1872.               }
1873.
1874.
1875.
1876.               // ================================================================
    =========
1877.               // Movie
1878.               @FXML
1879.               private void handleMovieSearch(ActionEvent event) {
1880.                   String movieSearch = txtMovieSearch.getText();
1881.                   boolean isSearchOK = checkMovieSearchForm(movieSearch);
1882.                   if(isSearchOK){
1883.                       // System.out.println("Search Ready");
1884.                       String searchMovieOf = cbMovieSearch.getValue();
1885.                       String searchMovie = searchMovieOf.replaceAll("\\s+","");
1886.                       //System.out.println(searchMovie.substring(0, 1).toLowerCase
    () + searchMovie.substring(1));
1887.                       cc.searchMovie(movieSearch, searchMovie.substring(0, 1).toLo
    werCase() + searchMovie.substring(1)); // search
1888.                       clearMovieSearchMessage();
1889.                       // clear list
1890.                       movieSearchList.clear();
1891.
1892.                       movieSearchListArray = cc.getMovieSearchList();
1893.                       try {
1894.                           for (Movie movie : movieSearchListArray) {
1895.                               movieSearchList.add(movie);
1896.                           }
1897.                       } catch (Exception e) {
1898.                           System.out.println(e);
1899.                       }
1900.                       if(movieSearchList.size() > 0){ // Found
1901.                           tbMovie.setItems(movieSearchList);
1902.                           msgMovieSearch.setText("Found!");
1903.                           msgMovieSearch.setTextFill(Color.rgb(21, 117, 84));
1904.                           btnCancleMovieSearch.setDisable(false);
```

```
1905.                           cancleMovieEdit(); // after search cancle edit
1906.                       }else{
1907.                           msgMovieSearch.setText("Not found!");
1908.                           msgMovieSearch.setTextFill(Color.rgb(210, 39, 30));
1909.                           btnCancleMovieSearch.setDisable(true);
1910.                           loadMovieData();
1911.                       }
1912.                   }else{
1913.                       loadMovieData();
1914.                   }
1915.               }
1916.               @FXML
1917.               private void handleCancleMovieSearch(ActionEvent event) {
1918.
1919.               // load normal table without search
1920.                   loadMovieData();
1921.                   clearMovieSearchMessage();
1922.                   txtMovieSearch.setText(null);
1923.                   btnCancleMovieSearch.setDisable(true);
1924.               }
1925.               @FXML
1926.               private void handleFilePoster(ActionEvent event) {
1927.                   // เปิด Dialog รับแต่ Image
1928.                   FileChooser fileChooser = new FileChooser();
1929.                   FileChooser.ExtensionFilter imageFilter = new FileChooser.Extens
    ionFilter("Image Files", "*.jpg", "*.png");
1930.                   fileChooser.getExtensionFilters().add(imageFilter);
1931.                   if (lastPath != null) {
1932.                      fileChooser.setInitialDirectory(lastPath);
1933.                   }
1934.                   selectedPoster = fileChooser.showOpenDialog(null);
1935.
1936.                   if (selectedPoster != null) {
1937.                      msgPoster.setText(selectedPoster.getName()); // Select file n
    ame
1938.                      msgPoster.setTextFill(Color.BLACK);
1939.                      InputStream image = getClass().getResourceAsStream("/cinema/p
    osterImage/"+selectedPoster.getName());
1940.                      posterImage = new Image(image);
1941.                      imagePoster.setImage(posterImage);
1942.                      lastPath = selectedPoster.getParentFile();
1943.                   }
1944.                   else {
1945.                      msgPoster.setText("");
1946.                      msgPoster.setTextFill(Color.BLACK);
1947.                      imagePoster.setImage(null);
1948.                      selectedPoster = null;
1949.                   }
1950.               }
1951.               @FXML
1952.               private void handleAddMovie(ActionEvent event) {
1953.                   String englishName = txtEnglishName.getText();
1954.                   String thaiName = txtThaiName.getText();
1955.                   String director = txtDirector.getText();
1956.                   String cast = txtCast.getText();
1957.                   String synopsis = txtSynopsis.getText();
1958.                   String genre = "";
1959.                   String hour = "";
1960.                   String minute = "";
1961.
1962.               String releaseDate = "";
1963.                   String poster = "";
1964.                   String trailer = txtTrailer.getText();
1965.
1966.                   boolean isMovieOK = checkMovieForm(englishName, thaiName, direct
    or, cast, synopsis, genre, hour, minute, releaseDate, poster, trailer);
```

```
1967.                   if(isMovieOK){
1968.                       List<Movie> checkMovieList = cc.getMovieList();
1969.                       boolean checkAlreadyExists = false;
1970.                       for (Movie movie : checkMovieList) {
1971.                           if(txtEnglishName.getText().equals(movie.getEnglishName(
    )) && !checkAlreadyExists) {
1972.                               checkAlreadyExists = true;
1973.                           }
1974.                       }
1975.                       if(checkAlreadyExists){
1976.                           AlertMaker.showErrorMessage("Add Movie Error", "This mov
    ie is already exists");
1977.                           clearMovieMessage();
1978.                           clearMovieForm();
1979.                       }
1980.                       else {
1981.                           System.out.println("Add movie");
1982.                           Movie movie = new Movie(
1983.                                   txtEnglishName.getText(),
1984.                                   txtThaiName.getText(),
1985.                                   txtDirector.getText(),
1986.                                   txtCast.getText(),
1987.                                   txtSynopsis.getText(),
1988.                                   cbGenre.getValue(),
1989.                                   cbTimeHour.getValue() + ":" +cbTimeMinute.getVal
    ue(),
1990.                                   convertDate(datePickerReleaseDate),
1991.                                   selectedPoster.getName(),
1992.                                   //selectedPoster.toURI().toString(),
1993.                                   txtTrailer.getText());
1994.
1995.
1996.                           cc.addMovie(movie);
1997.                           clearMovieMessage(); // clear label
1998.                           clearMovieForm(); // clear text form
1999.                           loadMovieData(); // load new list after add
2000.                           AlertMaker.showSimpleAlert("Add Complete", movie.toStrin
    g());
2001.                       }
2002.                   }
2003.
2004.           }
2005.               @FXML
2006.               private void handleSaveMovie(ActionEvent event) {
2007.                   String englishName = txtEnglishName.getText();
2008.                   String thaiName = txtThaiName.getText();
2009.                   String director = txtDirector.getText();
2010.                   String cast = txtCast.getText();
2011.                   String synopsis = txtSynopsis.getText();
2012.                   String genre = "";
2013.                   String hour = "";
2014.                   String minute = "";
2015.                   String releaseDate = "";
2016.                   String poster = "";
2017.                   String trailer = txtTrailer.getText();
2018.
2019.                   boolean isMovieOK = checkMovieForm(englishName, thaiName, direct
    or, cast, synopsis, genre, hour, minute, releaseDate, poster, trailer);
2020.                   System.out.println("Edit movie mode " + String.valueOf(editMovie
    Mode));
2021.
2022.                   List<Movie> checkMovieList = cc.getMovieList();
2023.                   boolean checkAlreadyExists = false;
2024.                   for (Movie movie : checkMovieList) {
2025.                       if(txtEnglishName.getText().equals(movie.getEnglishName()) &
    & !checkAlreadyExists) {
```

```
2026.                             if(selectedMovieEdit.getId() == movie.getId()) {
2027.                                 continue;
2028.                             }
2029.                             else {
2030.                                 checkAlreadyExists = true;
2031.                             }
2032.                         }
2033.                     }
2034.
2035.                     if(isMovieOK){
2036.                         System.out.println("Edit movie");
2037.                         Movie editMovie = new Movie(
2038.                                 txtEnglishName.getText(),
2039.                                 txtThaiName.getText(),
2040.                                 txtDirector.getText(),
2041.                                 txtCast.getText(),
2042.                                 txtSynopsis.getText(),
2043.                                 cbGenre.getValue(),
2044.                                 cbTimeHour.getValue() + ":" +cbTimeMinute.getValue()
    ,
2045.                                 convertDate(datePickerReleaseDate),
2046.
2047.                             selectedPoster.getName(),
2048.                                 // selectedPoster.toURI().toString(),
2049.                                 txtTrailer.getText());
2050.
2051.                         boolean isSame = checkMovieSameValue(editMovie);
2052.                         if(!isSame){
2053.                             if(checkAlreadyExists) {
2054.                                 AlertMaker.showErrorMessage("Save Movie Error", "Thi
    s Movie is already exists (English name is same with others)");
2055.                             }
2056.                             else {
2057.                                 System.out.println("Not same");
2058.                                 cc.editMovie(selectedMovieEdit.getId(), editMovie);

2059.                                 loadMovieData(); // load new list after add
2060.                                 // update edit select to new information
2061.                                 selectedMovieEdit = tbMovie.getItems().get(editMovie
    Id);
2062.                                 AlertMaker.showSimpleAlert("Saved!", "Save informati
    on completed");
2063.                                 cancleMovieEdit();
2064.                             }
2065.                         } else {
2066.                             AlertMaker.showSimpleAlert("No change!", "No changed for
     this movie information");
2067.                         }
2068.                     }
2069.
2070.                 }
2071.                 @FXML
2072.                 private void handleCancleEditMovie(ActionEvent event) {
2073.                     cancleMovieEdit();
2074.                 }
2075.                 @FXML
2076.                 private void handleMovieRefresh(ActionEvent event) {
2077.                     loadMovieData();
2078.                 }
2079.                 @FXML
2080.                 private void handleMovieEdit(ActionEvent event) throws ParseExceptio
    n {
2081.                         initMovieEditMode();
2082.                         System.out.println("Selected edit movie id : "+editMovieId);

2083.                 }
```

```
2084.               @FXML
2085.               private void handleMovieDelete(ActionEvent event) {
2086.                   Movie selectedMovieDelete = tbMovie.getSelectionModel().getSelec
    tedItem(); // เก็บมาเป็น object จาก list ที่เลือก
2087.
2088.               if(selectedMovieDelete == null){
2089.                       AlertMaker.showErrorMessage("No movie selected", "Please sel
    ect a movie for delete.");
2090.                       return;
2091.                   }
2092.
2093.                   Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
2094.                   alert.setTitle("Deleting movie");
2095.                   alert.setContentText("Are you sure want to delete the movie "+ s
    electedMovieDelete.getEnglishName()+ " ?");
2096.                   Optional<ButtonType> answer = alert.showAndWait();
2097.
2098.                   if(answer.get() == ButtonType.OK){
2099.                       cc.deleteMovie(selectedMovieDelete.getId());
2100.                       loadMovieData();
2101.                       cancleMovieEdit();
2102.                       clearMovieMessage();
2103.                   }
2104.               }
2105.               @FXML
2106.               private void handleMouseClickMovieEdit(MouseEvent event) throws Pars
    eException {
2107.                   if (event.isPrimaryButtonDown() && event.getClickCount() == 2) {

2108.                       initMovieEditMode();
2109.                       //System.out.println(selectedUserEdit);
2110.                       System.out.println("Selected edit movie id : "+editMovieId);

2111.                   }
2112.               }
2113.
2114.
2115.               // Promotion handle ----------------------------------------------
    -------
2116.               @FXML
2117.               private void handleAddPromotion(ActionEvent event) {
2118.                   String promotionName = txtPromotion.getText();
2119.                   String description = txtDescription.getText();
2120.                   String discount = txtDiscount.getText();
2121.                   String date = convertDate(pickerDate);
2122.                   System.out.println("Add promotion");
2123.                   System.out.println(date);
2124.
2125.                   boolean isAddable = checkPromotionForm(promotionName,description
    ,date,discount);
2126.
2127.                   double value_discount = Double.parseDouble(discount);
2128.                   //        double value_discount = 0.23;
2129.
2130.
2131.               if (value_discount < 0) {
2132.                       msgDiscount.setText("Discount cannot be under 0 !");
2133.                       msgDiscount.setTextFill(Color.rgb(210, 39, 30));
2134.                       isAddable = false;
2135.                   }
2136.
2137.                   List<Promotion> checkPromotionList = pc.getPromotionList();
2138.                   boolean isPromotionExists = false;
2139.                   for (Promotion promotion : checkPromotionList) {
2140.                       if(txtPromotion.getText().equals(promotion.getName()) && !is
    PromotionExists) {
```

```
2141.                             isPromotionExists = true;
2142.                         }
2143.                     }
2144.                     if(isAddable){
2145.                         if(isPromotionExists) {
2146.                             AlertMaker.showErrorMessage("Add Promotion Error","This
    promotion is already exists");
2147.                             clearPromotionMessage();
2148.                             clearPromotionForm();
2149.                         }
2150.                         else {
2151.                             Promotion newPromotion = new Promotion(promotionName, de
    scription,date , value_discount);
2152.                             pc.addPromotion(newPromotion);
2153.                             clearPromotionMessage(); // clear label
2154.                             clearPromotionForm(); // clear text form
2155.                             loadPromotionData(); // load new list after add
2156.                             AlertMaker.showSimpleAlert("OK",newPromotion.toString())
    ;
2157.                         }
2158.                     }
2159.                 }
2160.                 @FXML
2161.                 private void handleCanclePromotionSearch(ActionEvent event) {
2162.                     loadPromotionData();
2163.                     clearPromotionSearchMessage();
2164.                     txtPromotionSearch.setText(null);
2165.                     btnCancelPromotionSearch.setDisable(true);
2166.                 }
2167.                 @FXML
2168.                 private void handleSavePromotion(ActionEvent event) {
2169.                     String promotionName = txtPromotion.getText();
2170.                     String description = txtDescription.getText();
2171.                     String discount = txtDiscount.getText();
2172.                     String date = convertDate(pickerDate);
2173.
2174.
2175.                 double value_discount = Double.parseDouble(discount);
2176.
2177.                     boolean isAddable = checkPromotionForm(promotionName,description
    ,date,discount);
2178.
2179.                     List<Promotion> checkPromotionList = pc.getPromotionList();
2180.                     boolean isPromotionExists = false;
2181.                     for (Promotion promotion : checkPromotionList) {
2182.                         if(txtPromotion.getText().equals(promotion.getName()) && !is
    PromotionExists) {
2183.                             if(selectedPromotionEdit.getPromotionID() == promotion.g
    etPromotionID()) {
2184.                                 continue;
2185.                             }
2186.                             else {
2187.                                 isPromotionExists = true;
2188.                             }
2189.                         }
2190.                     }
2191.
2192.                     if (value_discount < 0) {
2193.                         msgDiscount.setText("Discount cannot be under 0 !");
2194.                         msgDiscount.setTextFill(Color.rgb(210, 39, 30));
2195.                         isAddable = false;
2196.                     }
2197.                     if(isAddable) {
2198.                         if(isPromotionExists) {
2199.                             AlertMaker.showErrorMessage("Save Promotion Error", "Thi
    s Promotion is already exists (Promotion name is same with others)");
```

```
2200.                       }
2201.                       else {
2202.                           System.out.println("Edit mode start...");
2203.                           Promotion editPromotion = new Promotion(promotionName, d
    escription,date , value_discount);
2204.                           pc.editPromotion(selectedPromotionEdit.getPromotionID(),
     editPromotion);
2205.                           cancelPromotionEdit();
2206.                       }
2207.                   }
2208.                   loadPromotionData();
2209.                   clearPromotionMessage();
2210.               }
2211.               @FXML
2212.               private void handleCancleEditPromotion(ActionEvent event) {
2213.                   cancelPromotionEdit();
2214.                   clearPromotionMessage();
2215.
2216.           }
2217.               @FXML
2218.               private void handlePromotionRefresh(ActionEvent event) {
2219.                   loadPromotionData();
2220.               }
2221.               @FXML
2222.               private void handlePromotionEdit(ActionEvent event) throws ParseExce
    ption{
2223.                   initPromotionEdit();
2224.               }
2225.               @FXML
2226.               private void handlePromotionDelete(ActionEvent event) {
2227.                   Promotion selectedPromotionDelete = tbPromotion.getSelectionMode
    l().getSelectedItem();
2228.                   if(selectedPromotionDelete == null) {
2229.                       AlertMaker.showErrorMessage("No promotion selected", "Please
     select a promotion to delete.");
2230.                       return;
2231.                   }
2232.
2233.                   Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
2234.                   alert.setTitle("Deleting Promotion");
2235.                   alert.setContentText("Are you sure want to delete the promotion
    "+ selectedPromotionDelete.getName()+ " ?");
2236.                   Optional<ButtonType> answer = alert.showAndWait();
2237.
2238.                   if(answer.get() == ButtonType.OK){
2239.                       pc.deletePromotion(selectedPromotionDelete.getPromotionID())
    ;
2240.                       cancelPromotionEdit();
2241.                       loadPromotionData();
2242.                   }
2243.               }
2244.               @FXML
2245.               private void handleMouseClickPromotionEdit(MouseEvent event) throws
    ParseException{
2246.                   if (event.isPrimaryButtonDown() && event.getClickCount() == 2) {

2247.                       initPromotionEdit();
2248.                   }
2249.               }
2250.               @FXML
2251.               private void handlePromotionSearch(ActionEvent event) {
2252.                   String promotionSearch = txtPromotionSearch.getText();
2253.                   boolean isSearch = checkPromotionSearchForm(promotionSearch);
2254.                   if(isSearch) {
2255.                       pc.searchPromotion(promotionSearch,"name");
2256.                       clearUserSearchMessage();
```

```
2257.
2258.                   promotionSearchList.clear();
2259.                       promotionSearchListArray = pc.getPromotionSearchList();
2260.                       try {
2261.                           for (Promotion promotion : promotionSearchListArray) {
2262.                               promotionSearchList.add(promotion);
2263.                           }
2264.                       } catch (Exception e) {
2265.                           System.out.println(e);
2266.                       }
2267.                       if(promotionSearchList.size() > 0){ // Found
2268.                           tbPromotion.setItems(promotionSearchList);
2269.                           msgPromotionSearch.setText("Found!");
2270.                           msgPromotionSearch.setTextFill(Color.rgb(21, 117, 84));

2271.                           btnCancelPromotionSearch.setDisable(false);
2272.                           cancelPromotionEdit(); // after search cancle edit
2273.                       }else{
2274.                           msgPromotionSearch.setText("Not found!");
2275.                           msgPromotionSearch.setTextFill(Color.rgb(210, 39, 30));

2276.                           btnCancelPromotionSearch.setDisable(true);
2277.                           loadPromotionData();
2278.                       }
2279.                   }
2280.                   else {
2281.                       System.out.println("Not found");
2282.                       loadPromotionData();
2283.                   }
2284.               }
2285.
2286.
2287.
2288.               // Theatre handle ----------------------------------------------------
    -----
2289.               @FXML
2290.               private void handleTheatreRefresh(ActionEvent event) {
2291.                   loadTheatreData();
2292.               }
2293.               @FXML
2294.               private void handleTheatreEdit(ActionEvent event)  {
2295.                   initTheatreEdit();
2296.               }
2297.               @FXML
2298.               private void handleTheatreDelete(ActionEvent event) {
2299.                   Theatre selectedTheatreDelete = tbTheatre.getSelectionModel().ge
    tSelectedItem();
2300.                   if(selectedTheatreDelete == null) {
2301.                       AlertMaker.showErrorMessage("No theatre selected", "Please s
    elect a theatre to delete.");
2302.
2303.                   return;
2304.                   }
2305.
2306.                   Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
2307.                   alert.setTitle("Deleting Theatre");
2308.                   alert.setContentText("Are you sure want to delete the theatre "+
     selectedTheatreDelete.getTheatreNumber()+ " ?");
2309.                   Optional<ButtonType> answer = alert.showAndWait();
2310.
2311.                   if(answer.get() == ButtonType.OK){
2312.                       cc.deleteTheatre(selectedTheatreDelete.getId());
2313.                       cancelTheatreEdit();
2314.                       loadTheatreData();
2315.                   }
2316.               }
```

```
2317.               @FXML
2318.               private void handleMouseClickTheatreEdit(MouseEvent event) {
2319.                   if (event.isPrimaryButtonDown() && event.getClickCount() == 2) {

2320.                       initTheatreEdit();
2321.                   }
2322.               }
2323.               @FXML
2324.               private void handleAddTheatre(ActionEvent event) {
2325.                   String theatreNum = txtTheatre.getText();
2326.
2327.                   boolean isAddable = checkTheatreForm(theatreNum);
2328.
2329.                   int theatre = Integer.parseInt(theatreNum);
2330.
2331.                   List<Theatre> checkTheatreExists = cc.getTheatreList();
2332.                   boolean isTheatreExists = false;
2333.                   for (Theatre checkTheatre : checkTheatreExists) {
2334.                       if((Integer.parseInt(txtTheatre.getText()) == (checkTheatre.
    getTheatreNumber())) && !isTheatreExists) {
2335.                           isTheatreExists = true;
2336.                       }
2337.                   }
2338.
2339.                   if(isAddable){
2340.                       if(isTheatreExists) {
2341.                           AlertMaker.showErrorMessage("Add Theatre Error", "This t
    heatre number is already exists");
2342.                           clearTheatreMessage();
2343.                           clearTheatreForm();
2344.                       }
2345.                       else {
2346.
2347.                           Theatre newTheatre = new Theatre(theatre,cbTheatreType.getVa
    lue());
2348.                           cc.addTheatre(newTheatre);
2349.                           clearTheatreMessage(); // clear label
2350.                           clearTheatreForm(); // clear text form
2351.                           loadTheatreData(); // load new list after add
2352.                       }
2353.                   }
2354.               }
2355.               @FXML
2356.               private void handleSaveTheatre(ActionEvent event) {
2357.                   String theatreNum = txtTheatre.getText();
2358.
2359.                   boolean isAddable = checkTheatreForm(theatreNum);
2360.                   int theatre = Integer.parseInt(theatreNum);
2361.                   Theatre editTheatre = new Theatre(theatre,cbTheatreType.getValue
    ());
2362.
2363.                   List<Theatre> checkTheatreExists = cc.getTheatreList();
2364.                   boolean isTheatreExists = false;
2365.                   for (Theatre checkTheatre : checkTheatreExists) {
2366.                       if((Integer.parseInt(txtTheatre.getText()) == (checkTheatre.
    getTheatreNumber())) && !isTheatreExists) {
2367.                           if(selectedTheatreEdit.getId() == checkTheatre.getId())
    {
2368.                               continue;
2369.                           }
2370.                           else {
2371.                               isTheatreExists = true;
2372.                           }
2373.                       }
2374.                   }
2375.
```

```
2376.                   if(isAddable) {
2377.                       if(isTheatreExists) {
2378.                           AlertMaker.showErrorMessage("Add Theatre Error", "This t
    heatre number is already exists");
2379.                           clearTheatreMessage();
2380.                       }
2381.                       else {
2382.                           cc.editTheatre(selectedTheatreEdit.getId(), editTheatre)
    ;
2383.                           cancelTheatreEdit();
2384.                       }
2385.                   }
2386.                   loadTheatreData();
2387.                   clearTheatreMessage();
2388.               }
2389.
2390.           @FXML
2391.               private void handleCancleEditTheatre(ActionEvent event) {
2392.                   cancelTheatreEdit();
2393.                   clearTheatreMessage();
2394.               }
2395.
2396.
2397.
2398.               // Showtime ============================================================
    ========
2399.
2400.               @FXML
2401.               private void handleAddShowtime(ActionEvent event) {
2402.                   String movieName = "";
2403.                   String incSp = txtIncreaseSeatPrice1.getText();
2404.                   String theatre = "";
2405.                   String hour = "";
2406.                   String min = "";
2407.                   String soundtrack = "";
2408.                   String date = "";
2409.
2410.                   boolean isShowtimeOK = checkShowtimeForm(movieName,theatre,incSp
    ,date,hour,min,soundtrack);
2411.
2412.
2413.                   if(isShowtimeOK){
2414.                       System.out.println("OK");
2415.                           System.out.println("Add showtime");
2416.                           String[] selectMovId = cbShowtimeSearch1.getValue().spli
    t(" :");
2417.                           String[] selectTheId = cbTheatreSearch1.getValue().split
    (" :");
2418.                           System.out.println("Movie ID: " + selectMovId[0]);
2419.                           System.out.println("Theatre ID: " + selectTheId[0]);
2420.                           selectMovId[0].replaceAll("\\s+","");
2421.                           selectTheId[0].replaceAll("\\s+","");
2422.
2423.                           int movie_id = Integer.valueOf(selectMovId[0]);
2424.                           int theatre_id = Integer.valueOf(selectTheId[0]);
2425.
2426.
2427.                           Movie m = cc.getMovie(movie_id);
2428.                           Theatre t = cc.getTheatre(theatre_id);
2429.                           String start = cbTimeHourShow1.getValue()+":"+cbTimeMinu
    teShow1.getValue();
2430.                           String sound = cbSoundtrack1.getValue();
2431.                           double incSeat = Double.parseDouble(incSp);
2432.                           String system = cbSystem.getValue();
2433.
2434.
```

```
2435.       //                  System.out.println(m.getEnglishName());
2436.
2437.           //                  System.out.println(t.getTheatreNumber());
2438.
2439.                           List<Showtime> listShowtime = cc.getShowtimeList();
2440.
2441.                            String myTime = m.getTime(); // เรียกระยะเวลาหนังที่จะสร้างมา , ถูกต้อง
2442.                            int newStartTimeHourInt = Integer.parseInt(cbTimeHourSho
    w1.getValue()); // หนังที่จะสร้าง
2443.                            int newStartTimeMinuteInt = Integer.parseInt(cbTimeMinut
    eShow1.getValue());
2444.                            int myTimeHour = Integer.parseInt(myTime.substring(0, 2)
    );
2445.                            int myTimeMinute = Integer.parseInt(myTime.substring(3))
    ;
2446.                            DecimalFormat twoDigit = new DecimalFormat("00");
2447.
2448.                            /* calculate end time */
2449.                            int newEndTimeHourInt = newStartTimeHourInt + myTimeHour
    ;
2450.                            int newEndTimeMinuteInt = newStartTimeMinuteInt + myTime
    Minute;
2451.                            if(newEndTimeMinuteInt >= 60) {
2452.                                newEndTimeHourInt = newEndTimeHourInt + 1;
2453.                                newEndTimeMinuteInt = newEndTimeMinuteInt - 60;
2454.                            }
2455.                            int newStartTime = Integer.parseInt(twoDigit.format(newS
    tartTimeHourInt) + "" + twoDigit.format(newStartTimeMinuteInt) + "");
2456.                            int newEndTime = Integer.parseInt(twoDigit.format(newEnd
    TimeHourInt) + "" + twoDigit.format(newEndTimeMinuteInt) + "");
2457.                             //&& showtime.getDate().equals(datePickerShowtime.getVa
    lue)
2458.
2459.                            boolean checkOverlap = false;
2460.                            DateTimeFormatter formatter = DateTimeFormatter.ofPatter
    n("d MMMM yyyy");
2461.
2462.                            for (Showtime showtime : listShowtime) {
2463.                                // โรงที่จะเพิ่ม ตรงกับเลขโรงไหมที่มีอยู่แล้วไหม
2464.                                if(showtime.getTheatre().getId() == theatre_id && (da
    tePickerShowtime.getValue().equals(LocalDate.parse(showtime.getDate(), formatter)))
     && (!checkOverlap)) {
2465.                                    String startString = showtime.getShowtime().subst
    ring(0, 2) + showtime.getShowtime().substring(3, 5);
2466.                                    String endString = showtime.getShowtime().substri
    ng(8, 10) + showtime.getShowtime().substring(11);
2467.
2468.                                int oldStartTime = Integer.parseInt(startString); //
    หนังที่มีอยู่แล้ว
2469.                                    int oldEndTime = Integer.parseInt(endString); //
    หนังที่มีอยู่แล้ว
2470.
2471.
2472.                                    // เวลาที่เริ่มที่จะเพิ่ม อยู่ในช่วง (เวลาที่มีอยู่) เวลาเริ่มและเวลาจบ
2473.                                    if(newStartTime >= oldStartTime && newStartTime <
     oldEndTime) {
2474.                                        // ชน
2475.                                        checkOverlap = true;
2476.                                    }
2477.                                    // เวลาเริ่มอยู่นอกช่วงนั้น
2478.                                    else {
2479.                                        // เวลาเริ่มน้อยกว่า แต่เวลาจบอยู่ในช่วง
2480.                                        if(newEndTime >= oldStartTime && newEndTime <
    = oldEndTime) {
```

```
2481.                                           // ชน
2482.                                           checkOverlap = true;
2483.                                       }
2484.                                   }
2485.                               }
2486.                           }
2487.
2488.                           if(!checkOverlap) {
2489.                               String showtimeDate = convertDate(datePickerShowtime
    );
2490.
2491.
2492.                               Showtime st = new Showtime(m,t,system,sound,showtime
    Date,start,incSeat);
2493.
2494.
2495.                               cc.addShowtime(st);
2496.                               clearShowtimeMessage(); // clear label
2497.                               clearShowtimeForm(); // clear text form
2498.                               loadShowtimeData(); // load new list after add
2499.                             //  AlertMaker.showSimpleAlert("Add Complete", st.toSt
    ring());
2500.                           }
2501.                           else {
2502.                               // Overlap
2503.                               AlertMaker.showSimpleAlert("Add Showtime Error", "Th
    e time is overlap.");
2504.                           }
2505.
2506.
2507.                   }else{
2508.                       System.out.println("NOT OK");
2509.                   }
2510.               }
2511.
2512.           @FXML
2513.               private void handleSaveShowtime(ActionEvent event) {
2514.                   String movieName = "";
2515.                   String incSp = txtIncreaseSeatPrice1.getText();
2516.                   String theatre = "";
2517.                   String hour = "";
2518.                   String min = "";
2519.                   String soundtrack = "";
2520.                   String date = "";
2521.
2522.                   boolean isShowtimeOK = checkShowtimeForm(movieName,theatre,incSp
    ,date,hour,min,soundtrack);
2523.
2524.
2525.                   if(isShowtimeOK){
2526.                       System.out.println("OK");
2527.                           System.out.println("Add showtime");
2528.                           String[] selectMovId = cbShowtimeSearch1.getValue().spli
    t(" :");
2529.                           String[] selectTheId = cbTheatreSearch1.getValue().split
    (" :");
2530.                           System.out.println("Movie ID: " + selectMovId[0]);
2531.                           System.out.println("Theatre ID: " + selectTheId[0]);
2532.                           selectMovId[0].replaceAll("\\s+","");
2533.                           selectTheId[0].replaceAll("\\s+","");
2534.
2535.                           int movie_id = Integer.valueOf(selectMovId[0]);
2536.                           int theatre_id = Integer.valueOf(selectTheId[0]);
2537.
2538.
2539.                           Movie m = cc.getMovie(movie_id);
```

```
2540.                           Theatre t = cc.getTheatre (theatre_id);
2541.
2542.                           String start = cbTimeHourShow1.getValue()+":"+cbTimeMinu
    teShow1.getValue();
2543.                           System.out.println("Start"+start);
2544.                           String sound = cbSoundtrack1.getValue();
2545.                           double incSeat = Double.parseDouble(incSp);
2546.
2547.                           String system = cbSystem.getValue();
2548.
2549.           //                  System.out.println(m.getEnglishName());
2550.           //                  System.out.println(t.getTheatreNumber());
2551.
2552.
2553.                           List<Showtime> listShowtime = cc.getShowtimeList();
2554.
2555.                           String myTime = m.getTime(); // เรียกระยะเวลาหนังที่จะสร้างมา , ถูกต้อง
2556.
2557.                       int newStartTimeHourInt = Integer.parseInt(cbTimeHourShow1.g
    etValue()); // หนังที่จะสร้าง
2558.                           int newStartTimeMinuteInt = Integer.parseInt(cbTimeMinut
    eShow1.getValue());
2559.                           int myTimeHour = Integer.parseInt(myTime.substring(0, 2)
    );
2560.                           int myTimeMinute = Integer.parseInt(myTime.substring(3))
    ;
2561.                           DecimalFormat twoDigit = new DecimalFormat("00");
2562.
2563.                           /* calculate end time */
2564.                           int newEndTimeHourInt = newStartTimeHourInt + myTimeHour
    ;
2565.                           int newEndTimeMinuteInt = newStartTimeMinuteInt + myTime
    Minute;
2566.                           if(newEndTimeMinuteInt >= 60) {
2567.                               newEndTimeHourInt = newEndTimeHourInt + 1;
2568.                               newEndTimeMinuteInt = newEndTimeMinuteInt - 60;
2569.                           }
2570.                           int newStartTime = Integer.parseInt(twoDigit.format(newS
    tartTimeHourInt) + "" + twoDigit.format(newStartTimeMinuteInt) + "");
2571.                           int newEndTime = Integer.parseInt(twoDigit.format(newEnd
    TimeHourInt) + "" + twoDigit.format(newEndTimeMinuteInt) + "");
2572.                            //&& showtime.getDate().equals(datePickerShowtime.getVa
    lue)
2573.
2574.                           boolean checkOverlap = false;
2575.                           DateTimeFormatter formatter = DateTimeFormatter.ofPatter
    n("d MMMM yyyy");
2576.
2577.                           for (Showtime showtime : listShowtime) {
2578.                               // โรงที่จะเพิ่ม ตรงกับเลขโรงไหมที่มีอยู่แล้วไหม
2579.                              if(showtime.getTheatre().getId() == theatre_id && (da
    tePickerShowtime.getValue().equals(LocalDate.parse(showtime.getDate(), formatter)))
     && (!checkOverlap)) {
2580.                                   String startString = showtime.getShowtime().subst
    ring(0, 2) + showtime.getShowtime().substring(3, 5);
2581.                                   String endString = showtime.getShowtime().substri
    ng(8, 10) + showtime.getShowtime().substring(11);
2582.                                   int oldStartTime = Integer.parseInt(startString);
     // หนังที่มีอยู่แแล้ว
2583.                                   int oldEndTime = Integer.parseInt(endString); //
    หนังที่มีอยู่แล้ว
2584.
2585.
2586.                               if(showtime.getId() == selectedShowtimeEdit.getId())
    {
```

```
2587.                                     continue;
2588.                                 }
2589.                                 else {
2590.                                     // เวลาที่เริ่มที่จะเพิ่ม อยู่ในช่วง (เวลาที่มีอยู่) เวลาเริ่มและเวลาจบ
2591.                                     if(newStartTime >= oldStartTime && newStartT
    ime < oldEndTime) {
2592.                                         // ชน
2593.                                         checkOverlap = true;
2594.                                     }
2595.                                     // เวลาเริ่มอยู่นอกช่วงนั้น
2596.                                     else {
2597.                                         // เวลาเริ่มน้อยกว่า แต่เวลาจบอยู่ในช่วง
2598.                                         if(newEndTime >= oldStartTime && newEndT
    ime <= oldEndTime) {
2599.                                             // ชน
2600.                                             checkOverlap = true;
2601.                                         }
2602.                                     }
2603.                                 }
2604.                             }
2605.                         }
2606.
2607.                         if(!checkOverlap) {
2608.                             String showtimeDate = convertDate(datePickerShowtime
    );
2609.
2610.
2611.                             Showtime st = new Showtime(m,t,system,sound,showtime
    Date,start,incSeat);
2612.                             cc.editShowtime(selectedShowtimeEdit.getId(),st);
2613.                             clearShowtimeMessage(); // clear label
2614.                             clearShowtimeForm(); // clear text form
2615.                             loadShowtimeData(); // load new list after add
2616.                             cancleShowtimeEdit();
2617.                           //  AlertMaker.showSimpleAlert("Add Complete", st.toSt
    ring());
2618.                           //  AlertMaker.showSimpleAlert("Add Complete", st.toSt
    ring());
2619.                         }
2620.                         else {
2621.                             // Overlap
2622.                             AlertMaker.showSimpleAlert("Save Showtime Error", "T
    he time is overlap.");
2623.                         }
2624.
2625.
2626.
2627.             }else{
2628.                     System.out.println("NOT OK");
2629.                 }
2630.
2631.
2632.
2633.
2634.             }
2635.             @FXML
2636.             private void handleCancleEditShowtime(ActionEvent event) {
2637.                 cancleShowtimeEdit();
2638.             }
2639.             @FXML
2640.             private void handleMouseClickShowtimeEdit(MouseEvent event) throws P
    arseException, IOException {
2641.                 if (event.isPrimaryButtonDown() && event.getClickCount() == 1) {

2642.                     initShowtimeEditMode();
```

```
2643.                       //System.out.println(selectedUserEdit);
2644.                       System.out.println("Selected edit Show id : "+editShowtimeId
    );
2645.                   }
2646.                   if (event.isPrimaryButtonDown() && event.getClickCount() == 2) {

2647.                       initShowtimeEditMode();
2648.                       cc.setSelectShowtime(selectedShowtimeEdit.getId());
2649.                       Parent root = FXMLLoader.load(getClass().getResource("/cinem
    a/admin/ViewSeat.fxml"));
2650.                       Stage stage = new Stage(StageStyle.DECORATED);
2651.                       stage.setTitle("Seat List");
2652.                       stage.setScene(new Scene(root));
2653.                       //stage.setMaximized(true);
2654.                       stage.show();
2655.                   }
2656.               }
2657.               @FXML
2658.               private void handleShowtimeRefresh(ActionEvent event) {
2659.                   loadShowtimeData();
2660.                   clearShowtimeForm();
2661.                   clearShowtimeMessage();
2662.               }
2663.               @FXML
2664.               private void handleShowtimeEdit(ActionEvent event) throws ParseExcep
    tion {
2665.                   initShowtimeEditMode();
2666.               }
2667.               @FXML
2668.               private void handleShowtimeDelete(ActionEvent event) {
2669.
2670.               Showtime selectedShowtimeDelete = tbShowtime1.getSelectionModel().ge
    tSelectedItem(); // เก็บมาเป็น object จาก list ที่เลือก
2671.
2672.                   if(selectedShowtimeDelete == null){
2673.                       AlertMaker.showErrorMessage("No Showtime selected", "Please
    select a Showtime for delete.");
2674.                       return;
2675.                   }
2676.
2677.                   Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
2678.                   alert.setTitle("Deleting Showtime");
2679.                   alert.setContentText("Are you sure want to delete the Showtime I
    D "+ selectedShowtimeDelete.getId()+ " ?");
2680.                   Optional<ButtonType> answer = alert.showAndWait();
2681.
2682.                   if(answer.get() == ButtonType.OK){
2683.                       System.out.println(selectedShowtimeDelete.getId());
2684.                       cc.deleteShowtime(selectedShowtimeDelete.getId());
2685.                       System.out.println("Deleting Showtime..... ");
2686.                       loadShowtimeData();
2687.                       cancleShowtimeEdit();
2688.                       clearShowtimeMessage();
2689.                   }
2690.               }
2691.               @FXML
2692.               private void handleFindShowtimeSystem(ActionEvent event) {
2693.                   System.out.println("Load System");
2694.                   if(editShowtime){
2695.                       changeTheatre = true;
2696.                   }
2697.                   if(cbTheatreSearch1.getValue() != null){
2698.                       loadSystemCombo();
2699.                       cbSystem.setDisable(false);
2700.                   }
2701.
```

```
2702.               }
2703.
2704.               @FXML
2705.               private void handleRefreshBooking(ActionEvent event) {
2706.                   initBookingCol();
2707.                   loadBookingData();
2708.                   setBookingTotal();
2709.               }
2710.
```

Class AdminMainLoader

```
1.  package cinema.admin;
2.
3.
4.      import cinema.ui.auth.*;
5.      import javafx.application.Application;
6.      import static javafx.application.Application.launch;
7.      import javafx.fxml.FXMLLoader;
8.      import javafx.scene.Parent;
9.      import javafx.scene.Scene;
10.     import javafx.stage.Stage;
11.     import cinema.ui.CinemaUtility;
12.
13.
14.     public class AdminMainLoader extends Application{
15.         @Override
16.         public void start(Stage stage) throws Exception {
17.             Parent root = FXMLLoader.load(getClass().getResource("/cinema/admin/Adm
    inMain.fxml"));
18.             Scene scene = new Scene(root);
19.             stage.setScene(scene);
20.             stage.setTitle("KMITL Cinema");
21.             stage.setMaximized(true);
22.             stage.show();
23.             CinemaUtility.setStageIcon(stage);
24.         }
25.
26.
27.         public static void main(String[] args) {
28.             launch(args);
29.         }
30.     }
```

Class ViewSeatController

```
1.  package cinema.admin;
2.
3.
4.      import cinema.CinemaController;
5.      import cinema.Seat;
6.      import cinema.Showtime;
7.      import cinema.User;
8.      import com.jfoenix.controls.JFXListView;
9.      import java.net.URL;
10.     import java.text.Collator;
11.     import java.util.ArrayList;
12.     import java.util.List;
13.     import java.util.ResourceBundle;
14.     import javafx.collections.FXCollections;
15.     import javafx.collections.ObservableList;
16.     import javafx.event.ActionEvent;
17.     import javafx.fxml.FXML;
18.     import javafx.fxml.Initializable;
19.     import javafx.scene.control.MenuItem;
20.     import javafx.scene.control.TableColumn;
21.     import javafx.scene.control.TableView;
22.     import javafx.scene.control.cell.PropertyValueFactory;
23.     import javafx.scene.input.MouseEvent;
24.     import javafx.scene.text.Text;
25.
26.
27.     /**
28.      * FXML Controller class
29.      *
30.      * @author Phijak
31.      */
32.     public class ViewSeatController implements Initializable {
33.         private CinemaController cc;
34.         @FXML
35.         private TableView<Seat> tbSeat;
36.         @FXML
37.         private TableColumn<Seat, Integer> colSeatId;
38.         @FXML
39.         private TableColumn<Seat, String> colSeatName;
40.         @FXML
41.         private TableColumn<Seat, Double> colSeatPrice;
42.         @FXML
43.         private TableColumn<Seat, Boolean> colSeatStatus;
44.         @FXML
45.         private TableColumn<Seat, String> colSeatType;
46.
47.
48.         public ViewSeatController() {
49.             this.cc = cc.getInstance();
50.         }
51.         @Override
52.         public void initialize(URL url, ResourceBundle rb) {
53.             //ObservableList<String> seatData = FXCollections.observableArrayList()
    ;
54.
55.             ObservableList<Seat> seatList = FXCollections.observableArrayList();
56.             colSeatId.setCellValueFactory(new PropertyValueFactory<>("id"));
57.             colSeatName.setCellValueFactory(new PropertyValueFactory<>("seatName"))
    ;
58.             colSeatPrice.setCellValueFactory(new PropertyValueFactory<>("seatPrice"
    ));
```

```
59.              colSeatStatus.setCellValueFactory(new PropertyValueFactory<>("seatStatu
   s"));
60.              colSeatType.setCellValueFactory(new PropertyValueFactory<>("seatType"))
   ;
61.
62.             seatList.clear();
63.              System.out.println(cc.getSelectShowtime());
64.             Showtime s = cc.getShowtime(cc.getSelectShowtime());
65.             List<Seat> seatShowtime = s.getSeatList();
66.             try{
67.                 for (Seat seat : seatShowtime) {
68.                  seatList.add(seat);
69.                  //seatData.add(seat.toString());
70.                 }
71.             } catch (Exception e) {
72.                  System.out.println(e);
73.             }
74.             tbSeat.setItems(seatList);
75.             tbSeat.getSortOrder().add(colSeatName);
76.
77.
78.             //seatListview.getItems().setAll(seatData);
79.
80.         }
81.     }
```

ส่วนUI

Class AlertMaker

```
1.  package cinema.ui;
2.
3.
4.      import java.io.PrintWriter;
5.      import java.io.StringWriter;
6.      import javafx.scene.control.Alert.AlertType;
7.      import javafx.scene.control.Alert;
8.      import javafx.scene.control.Label;
9.      import javafx.scene.control.TextArea;
10.     import javafx.scene.layout.GridPane;
11.     import javafx.scene.layout.Priority;
12.
13.
14.     public class AlertMaker {
15.
16.
17.         public static void showSimpleAlert(String title, String content) {
18.             Alert alert = new Alert(AlertType.INFORMATION);
19.             alert.setTitle(title);
20.             alert.setHeaderText(null);
21.             alert.setContentText(content);
22.
23.
24.             alert.showAndWait();
25.         }
26.
27.
28.         public static void showErrorMessage(String title, String content) {
29.             Alert alert = new Alert(AlertType.ERROR);
30.             alert.setTitle("Error");
31.             alert.setHeaderText(title);
32.             alert.setContentText(content);
33.
34.
35.             alert.showAndWait();
36.         }
37.
38.
39.         public static void showErrorMessage(Exception ex) {
40.             Alert alert = new Alert(AlertType.ERROR);
41.             alert.setTitle("Error occured");
42.             alert.setHeaderText("Error Occured");
43.             alert.setContentText(ex.getLocalizedMessage());
44.
45.
46.             StringWriter sw = new StringWriter();
47.             PrintWriter pw = new PrintWriter(sw);
48.             ex.printStackTrace(pw);
49.             String exceptionText = sw.toString();
50.
51.
52.
53.             Label label = new Label("The exception stacktrace was:");
54.
55.
56.             TextArea textArea = new TextArea(exceptionText);
57.             textArea.setEditable(false);
58.             textArea.setWrapText(true);
59.
```

```
60.
61.             textArea.setMaxWidth(Double.MAX_VALUE);
62.             textArea.setMaxHeight(Double.MAX_VALUE);
63.             GridPane.setVgrow(textArea, Priority.ALWAYS);
64.             GridPane.setHgrow(textArea, Priority.ALWAYS);
65.
66.
67.             GridPane expContent = new GridPane();
68.             expContent.setMaxWidth(Double.MAX_VALUE);
69.             expContent.add(label, 0, 0);
70.             expContent.add(textArea, 0, 1);
71.
72.
73.             alert.getDialogPane().setExpandableContent(expContent);
74.             alert.showAndWait();
75.         }
76.
77.          public static void showErrorMessage(Exception ex, String title, String con
    tent) {
78.             Alert alert = new Alert(AlertType.ERROR);
79.             alert.setTitle("Error occured");
80.             alert.setHeaderText(title);
81.             alert.setContentText(content);
82.
83.
84.             StringWriter sw = new StringWriter();
85.             PrintWriter pw = new PrintWriter(sw);
86.             ex.printStackTrace(pw);
87.             String exceptionText = sw.toString();
88.
89.
90.             Label label = new Label("The exception stacktrace was:");
91.
92.
93.             TextArea textArea = new TextArea(exceptionText);
94.             textArea.setEditable(false);
95.             textArea.setWrapText(true);
96.
97.
98.             textArea.setMaxWidth(Double.MAX_VALUE);
99.             textArea.setMaxHeight(Double.MAX_VALUE);
100.                GridPane.setVgrow(textArea, Priority.ALWAYS);
101.                GridPane.setHgrow(textArea, Priority.ALWAYS);
102.
103.
104.                GridPane expContent = new GridPane();
105.                expContent.setMaxWidth(Double.MAX_VALUE);
106.                expContent.add(label, 0, 0);
107.                expContent.add(textArea, 0, 1);
108.
109.
110.                alert.getDialogPane().setExpandableContent(expContent);
111.                alert.showAndWait();
112.            }
113.        }
```

Class CinemaUtility

```
1.  package cinema.ui;
2.
3.
4.      import com.jfoenix.controls.JFXPasswordField;
5.      import com.jfoenix.controls.JFXTextField;
6.      import javafx.scene.control.Label;
7.      import javafx.scene.image.Image;
8.      import javafx.stage.Stage;
9.
10.
11.     public class CinemaUtility {
12.         private static final String IMAGE_LOC = "/cinema/ui/images/logo.png";
13.
14.         public static void setStageIcon(Stage stage){
15.             stage.getIcons().add(new Image(IMAGE_LOC));
16.         }
17.
18.         public static void resetTextStyle(JFXTextField tf, Label lb, String name){

19.             tf.getStyleClass().removeIf(style -> style.equals(name));
20.             lb.setText("");
21.         }
22.
23.         public static void resetTextStyle(JFXPasswordField tf, Label lb,String name
    ){
24.             tf.getStyleClass().removeIf(style -> style.equals(name));
25.             lb.setText("");
26.         }
27.     }
```

Class FieldValidation

```
1.  package cinema.ui;
2.
3.
4.      import com.jfoenix.controls.JFXPasswordField;
5.      import com.jfoenix.controls.JFXTextField;
6.      import java.util.regex.Matcher;
7.      import java.util.regex.Pattern;
8.      import javafx.scene.control.Label;
9.
10.
11.
12.
13.     public class FieldValidation {
14.         public static boolean isTextFieldNotEmpty(JFXTextField textField){
15.             boolean b = false;
16.             if(!(textField.getText().isEmpty())){
17.                 b = true;
18.             }
19.             return b;
20.         }
21.
22.         public static boolean isTextFieldNotEmpty(JFXPasswordField textField){
23.             boolean b = false;
24.             if(!(textField.getText().isEmpty())){
25.                 b = true;
26.             }
27.             return b;
28.         }
29.
30.         public static boolean isTextNotEmpty(String string){
31.             boolean b = false;
32.             if(!(string.isEmpty())){
33.                 b = true;
34.             }
35.             return b;
36.         }
37.
38.         public static boolean isTextNotEmpty(String string,Label label, String erro
    rMessage){
39.             boolean b = true;
40.             if(!isTextNotEmpty(string)){
41.                 b = false;
42.                 // ให้ Label แสดงข้อความ และเปลี่ยนเป็นสี Error
43.                 label.setText(errorMessage);
44.                 label.getStyleClass().add("text-error");
45.
46.
47.             // ให้ Textfield เปลี่ยนเป็นสี Error
48.                 //textField.getStyleClass().add("text-field-error");
49.                 // AlertMaker.showErrorMessage("Form field can't be empty","Please
    enter ");
50.             }else{
51.                 b = true;
52.             }
53.
54.             return b;
55.         }
56.
57.         public static boolean isTextFieldNotEmpty(JFXTextField textField,Label labe
    l, String errorMessage){
58.             boolean b = true;
59.             if(!isTextFieldNotEmpty(textField)){
```

```
60.                 b = false;
61.                 // ให้ Label แสดงข้อความ และเปลี่ยนเป็นสี Error
62.                 label.setText(errorMessage);
63.                 label.getStyleClass().add("text-error");
64.
65.                 // ให้ Textfield เปลี่ยนเป็นสี Error
66.                 textField.getStyleClass().add("text-field-error");
67.                 // AlertMaker.showErrorMessage("Form field can't be empty","Please
    enter ");
68.             }else{
69.                 b = true;
70.             }
71.
72.             return b;
73.         }
74.
75.         public static boolean isTextFieldNotEmpty(JFXPasswordField textField,Label
    label, String errorMessage){
76.             boolean b = true;
77.             if(!isTextFieldNotEmpty(textField)){
78.                 b = false;
79.                 // ให้ Label แสดงข้อความ และเปลี่ยนเป็นสี Error
80.                 label.setText(errorMessage);
81.                 label.getStyleClass().add("text-error");
82.
83.                 // ให้ Textfield เปลี่ยนเป็นสี Error
84.                 textField.getStyleClass().add("text-field-error");
85.                 // AlertMaker.showErrorMessage("Form field can't be empty","Please
    enter ");
86.             }else{
87.                 b = true;
88.
89.         }
90.
91.             return b;
92.         }
93.
94.         public static boolean isTextMatch(JFXPasswordField text, JFXPasswordField c
    onfirmText){
95.             boolean b = false;
96.             if(text.getText().equals(confirmText.getText())){
97.                 b = true;
98.             }
99.             return b;
100.             }
101.
102.             public static boolean isTextMatch(String text, JFXTextField confirmT
    ext){
103.                 boolean b = false;
104.                 if(text.equals(confirmText.getText())){
105.                     b = true;
106.                 }
107.                 return b;
108.             }
109.
110.             public static boolean isTextMatch(JFXPasswordField text, JFXPassword
    Field confirmText,Label label, String errorMessage){
111.                 boolean b = true;
112.                 if(!isTextMatch(text, confirmText)){
113.                     b = false;
114.                     // ให้ Label แสดงข้อความ และเปลี่ยนเป็นสี Error
115.                     label.setText(errorMessage);
116.                     label.getStyleClass().add("text-error");
117.
118.                     // ให้ Textfield เปลี่ยนเป็นสี Error
```

```
                        confirmText.getStyleClass().add("text-field-
error"); // บอก Confirm password ไม่ตรง
                    }else{
                        b = true;
                    }
                    return b;
                }

                public static boolean isTextMatch(String text, JFXTextField confirmT
ext,Label label, String errorMessage){
                    boolean b = true;
                    if(!isTextMatch(text, confirmText)){
                        b = false;
                        // ให้ Label แสดงข้อความ และเปลี่ยนเป็นสี Error
                        label.setText(errorMessage);
                        label.getStyleClass().add("text-error");

                        // ให้ Textfield เปลี่ยนเป็นสี Error
                        confirmText.getStyleClass().add("text-field-
error"); // บอก Confirm password ไม่ตรง
                    }else{
                        b = true;
                    }
                    return b;
                }

                public static boolean isValidEmailAddress(String email, JFXTextField
 confirmText,Label label, String errorMessage) {
                       boolean b = true;
                       String ePattern = "^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@((\\[[0-
9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\])|(([a-zA-Z\\-0-9]+\\.)+[a-zA-
Z]{2,}))$";
                       Pattern p = java.util.regex.Pattern.compile(ePattern);
                       Matcher m = p.matcher(email);
                       if(!m.matches()){
                           b = false;
                           label.setText(errorMessage);
                           label.getStyleClass().add("text-error");

                           // ให้ Textfield เปลี่ยนเป็นสี Error
                           confirmText.getStyleClass().add("text-field-
error"); // บอก Confirm password ไม่ตรง
                       }else{
                           b = true;
                       }
                       return b;
                }
            }
```

หน้า authentication

Class LoginController

```
1.  package cinema.ui.auth;
2.
3.  import cinema.CinemaController;
4.  import cinema.User;
5.  import cinema.UserController;
6.  import cinema.ui.AlertMaker;
7.  import cinema.ui.CinemaUtility;
8.  import cinema.ui.FieldValidation;
9.  import com.jfoenix.controls.JFXPasswordField;
10. import com.jfoenix.controls.JFXTextField;
11. import java.io.IOException;
12. import java.util.List;
13. import javafx.fxml.FXMLLoader;
14. import javafx.scene.Node;
15. import javafx.scene.Parent;
16. import javafx.scene.Scene;
17. import javafx.scene.control.Label;
18. import javafx.scene.input.MouseEvent;
19. import javafx.scene.layout.AnchorPane;
20. import javafx.stage.Stage;
21. import javafx.stage.StageStyle;
22. import java.net.URL;
23. import java.util.ResourceBundle;
24. import javafx.event.ActionEvent;
25. import javafx.fxml.FXML;
26. import javafx.fxml.Initializable;
27. import javafx.geometry.Rectangle2D;
28. import javafx.scene.paint.Color;
29. import javafx.stage.Screen;
30.
31. public class LoginController implements Initializable {
32.
33.     @FXML
34.     private AnchorPane rootPane;
35.     @FXML
36.     private JFXTextField txtUsername;
37.     @FXML
38.     private Label labelUsername;
39.     @FXML
40.     private JFXPasswordField txtPassword;
41.     @FXML
42.     private Label labelPassword;
43.
44.     List<User> allUser;
45.     UserController uc;
46.     CinemaController cc;
47.     User loginUser;
48.
49.
50.     public LoginController() {
51.         this.uc = uc.getInstance();
52.         this.cc = cc.getInstance();
53.     }
54.
55.     // ลบค่า Label
56.     public void clearLabel(){
57.         labelUsername.setText("");
58.         labelPassword.setText("");
59.     }
```

```
60.
61.     // ลบค่า Form
62.     public void clearForm(){
63.         txtUsername.setText("");
64.         txtPassword.setText("");
65.     }
66.
67.     public void resetAllStyle(){
68.         resetUsernameStyle();
69.         resetPasswordStyle();
70.     }
71.
72.     public void resetUsernameStyle(){
73.         CinemaUtility.resetTextStyle(txtUsername, labelUsername, "text-field-
    error");
74.     }
75.
76.     public void resetPasswordStyle(){
77.         CinemaUtility.resetTextStyle(txtPassword, labelPassword , "text-field-
    error");
78.     }
79.
80.     @Override
81.     public void initialize(URL url, ResourceBundle rb) {
82.         clearForm(); // clear initial form
83.         clearLabel(); // clear initial label
84. //         cinema.getInstance();
85.     }
86.
87.     @FXML
88.     private void handleLogin(ActionEvent event) throws IOException {
89.         boolean isUernameNotEmpty = FieldValidation.isTextFieldNotEmpty(txtUsername
    , labelUsername, "ต้องใส่ชื่อผู้ใช้งาน");
90.         boolean isPasswordNotEmpty = FieldValidation.isTextFieldNotEmpty(txtPasswor
    d, labelPassword, "ต้องใส่รหัสผ่าน");
91.
92.         if(isUernameNotEmpty && isPasswordNotEmpty){
93.             System.out.println("Login...");
94.             resetAllStyle();
95.
96.             if(uc.checkExistUsername(txtUsername.getText())) {
97.                 // Have this username in database
98.                 if (uc.checkValidUser(txtUsername.getText(), txtPassword.getText())
    ){
99.                     loginUser = uc.getLoginUser();
100.                        //AlertMaker.showSimpleAlert("Login Completed", loginUse
    r.toString());
101.            //                closeStage(); // close screen
102.            //                loadMain(); // show main screen
103.                        System.out.println(loginUser.getType());
104.            //                Stage window = (Stage)((Node)event.getSource()).getSce
    ne().getWindow();
105.
106.                        if(loginUser.getType().equals("Admin")){
107.                            // Go to admin
108.                            System.out.println("admin");
109.            //                loadScreen(window , "/cinema/admin/AdminMain.fxml"
    );
110.                            cc.setIsSelectedSeat(false);
111.                            Parent parent;
112.                            parent = FXMLLoader.load(getClass().getResource("/ci
    nema/admin/AdminMain.fxml"));
113.                            Scene parentScene = new Scene(parent);
114.                            Stage window = (Stage)((Node)event.getSource()).getS
    cene().getWindow();
```

```
115.                                 window.setScene(parentScene);
116.                                 window.show();
117.                             }
118.                             else{
119.                                 // Goto staff or user
120.                                 System.out.println("user");
121.                                 if(loginUser.getType().equals("Staff")) {
122.                                     if(cc.isSelectedSeat()) {
123.                                         cc.setIsSelectedSeat(false);
124.
125.                                         Parent parent;
126.                                         parent = FXMLLoader.load(getClass().getResou
    rce("/cinema/ui/summaryStaff/summaryStaff.fxml"));
127.                                         Scene parentScene = new Scene(parent);
128.                                         Stage window = (Stage)((Node)event.getSource
    ()).getScene().getWindow();
129.                                         window.setScene(parentScene);
130.                                         window.show();
131.                                     }
132.                                     else {
133.                 //                          loadScreen(window , "/cinema/ui/showmovie/
    showmovie.fxml");
134.                                         Parent parent;
135.                                         parent = FXMLLoader.load(getClass().getResou
    rce("/cinema/ui/showmovie/showmovie.fxml"));
136.                                         Scene parentScene = new Scene(parent);
137.                                         Stage window = (Stage)((Node)event.getSource
    ()).getScene().getWindow();
138.                                         window.setScene(parentScene);
139.                                         window.show();
140.                                     }
141.                                 }
142.                                 else {
143.                                     if(cc.isSelectedSeat()) {
144.                 //                          loadScreen(window , "/cinema/ui/summary/su
    mmary.fxml");
145.                                         cc.setIsSelectedSeat(false);
146.                                         Parent parent;
147.                                         parent = FXMLLoader.load(getClass().getResou
    rce("/cinema/ui/summary/summary.fxml"));
148.                                         Scene parentScene = new Scene(parent);
149.                                         Stage window = (Stage)((Node)event.getSource
    ()).getScene().getWindow();
150.                                         window.setScene(parentScene);
151.                                         window.show();
152.                                     }
153.                                     else {
154.                 //                          loadScreen(window , "/cinema/ui/showmovie/
    showmovie.fxml");
155.                                         Parent parent;
156.                                         parent = FXMLLoader.load(getClass().getResou
    rce("/cinema/ui/showmovie/showmovie.fxml"));
157.                                         Scene parentScene = new Scene(parent);
158.                                         Stage window = (Stage)((Node)event.getSource
    ()).getScene().getWindow();
159.                                         window.setScene(parentScene);
160.                                         window.show();
161.                                     }
162.                                 }
163.                             }
164.                         } else {
165.                             labelPassword.setTextFill(Color.RED);
166.                             labelPassword.setText("รหัสผ่านไม่ถูกต้อง");
167.                             txtPassword.setText("");
168.                             txtPassword.requestFocus();
169.                         }
```

```
170.                     }
171.                     else {
172.                         AlertMaker.showErrorMessage("Login Failed", "ไม่พบชื่อผู้ใช้งานนี้อยู่ใน
    ฐานข้อมูล");
173.                         clearLabel();
174.                         clearForm();
175.                         txtUsername.requestFocus();
176.                     }
177.
178.
179.                 }else{
180.                     if(isUernameNotEmpty){
181.                         resetUsernameStyle(); // if username correct remove error st
    yle
182.                     }
183.                     if(isPasswordNotEmpty){
184.                         resetPasswordStyle(); // if password correct remove error st
    yle
185.                     }
186.
187.                 }
188.             }
189.
190.             @FXML
191.             private void handleRegister(ActionEvent event){
192.                 Stage window = (Stage)((Node)event.getSource()).getScene().getWindow
    ();
193.                 loadScreen(window , "/cinema/ui/auth/Register.fxml");
194.             }
195.
196.             public void loadScreen(Stage window , String path){
197.                 try {
198.                     ///((Stage)rootPane.getScene().getWindow()).close();
199.                     Parent parent;
200.                     parent = FXMLLoader.load(getClass().getResource(path));
201.                     Scene parentScene = new Scene(parent);
202.                     window.setScene(parentScene);
203.                     window.show();
204.                 } catch (IOException ex) {
205.                     System.out.println(ex);
206.                 }
207.             }
208.
209.             @FXML
210.             private void handleMain(ActionEvent event) {
211.                 Stage window = (Stage)((Node)event.getSource()).getScene().getWindow
    ();
212.                 loadScreen(window , "/cinema/ui/showmovie/showmovie.fxml");
213.             }
214.         }
```

Class LoginLoader

```
1.  package cinema.ui.auth;
2.
3.
4.      import javafx.application.Application;
5.      import static javafx.application.Application.launch;
6.      import javafx.fxml.FXMLLoader;
7.      import javafx.scene.Parent;
8.      import javafx.scene.Scene;
9.      import javafx.stage.Stage;
10.     import cinema.ui.CinemaUtility;
11.
12.
13.     public class LoginLoader extends Application{
14.         @Override
15.         public void start(Stage stage) throws Exception {
16.             Parent root = FXMLLoader.load(getClass().getResource("/cinema/ui/auth/L
    ogin.fxml"));
17.             Scene scene = new Scene(root);
18.             scene.getStylesheets().add("/cinema/ui/theme.css");
19.             stage.setScene(scene);
20.             stage.setTitle("KMITL Cinema");
21.             //stage.setMaximized(true);
22.             stage.show();
23.             CinemaUtility.setStageIcon(stage);
24.         }
25.
26.
27.         public static void main(String[] args) {
28.             launch(args);
29.         }
30.     }
```

Class RegisterController

```
1.  package cinema.ui.auth;
2.
3.
4.      import cinema.User;
5.      import cinema.UserController;
6.      import cinema.ui.AlertMaker;
7.      import cinema.ui.CinemaUtility;
8.      import cinema.ui.FieldValidation;
9.      import com.jfoenix.controls.JFXPasswordField;
10.     import com.jfoenix.controls.JFXTextField;
11.     import java.io.IOException;
12.     import java.net.URL;
13.     import java.util.List;
14.     import java.util.ResourceBundle;
15.     import javafx.event.ActionEvent;
16.     import javafx.fxml.FXML;
17.     import javafx.fxml.FXMLLoader;
18.     import javafx.fxml.Initializable;
19.     import javafx.geometry.Rectangle2D;
20.     import javafx.scene.Node;
21.     import javafx.scene.Parent;
22.     import javafx.scene.Scene;
23.     import javafx.scene.control.Label;
24.     import javafx.scene.layout.AnchorPane;
25.     import javafx.stage.Screen;
26.     import javafx.stage.Stage;
27.
28.
29.     public class RegisterController implements Initializable {
30.
31.
32.         @FXML
33.         private AnchorPane rootPane;
34.         @FXML
35.         private JFXTextField txtUsername;
36.         @FXML
37.         private Label labelUsername;
38.         @FXML
39.         private JFXPasswordField txtPassword;
40.         @FXML
41.         private Label labelPassword;
42.         @FXML
43.         private JFXPasswordField txtConfirmPassword;
44.         @FXML
45.         private Label labelConfirmPassword;
46.         @FXML
47.
48.     private JFXTextField txtFirstName;
49.         @FXML
50.         private Label labelFirstName;
51.         @FXML
52.         private JFXTextField txtLastName;
53.         @FXML
54.         private Label labelLastName;
55.         @FXML
56.         private JFXTextField txtEmail;
57.         @FXML
58.         private Label labelEmail;
59.
60.
61.         List<User> allUser;
62.         UserController uc;
```

```
63.
64.         public RegisterController() {
65.             this.uc = uc.getInstance();
66.         }
67.
68.         @Override
69.         public void initialize(URL location, ResourceBundle resources) {
70.             clearLabel();
71.             uc.getInstance();
72.         }
73.
74.
75.         @FXML
76.         private void handleCreate(ActionEvent event) throws IOException {
77.             // Form validation
78.             boolean isUernameNotEmpty = FieldValidation.isTextFieldNotEmpty(txtUser
    name, labelUsername, "ใส่ชื่อผู้ใช้งาน");
79.             boolean isPasswordNotEmpty = FieldValidation.isTextFieldNotEmpty(txtPas
    sword, labelPassword, "ใส่รหัสผ่าน");
80.             boolean isConfirmPasswordNotEmpty = FieldValidation.isTextFieldNotEmpty
    (txtConfirmPassword, labelConfirmPassword, "ใส่ยืนยันรหัสผ่าน");
81.             boolean isFirstNameNotEmpty = FieldValidation.isTextFieldNotEmpty(txtFi
    rstName, labelFirstName, "ใส่ชื่อจริง");
82.             boolean isLastNameNotEmpty = FieldValidation.isTextFieldNotEmpty(txtLas
    tName, labelLastName, "ใส่นามสกุล");
83.             boolean isEmailNotEmpty = FieldValidation.isTextFieldNotEmpty(txtEmail,
     labelEmail, "ใส่อีเมล");
84.
85.
86.         if(isUernameNotEmpty && isPasswordNotEmpty && isConfirmPasswordNotEmpty &&
    isFirstNameNotEmpty && isLastNameNotEmpty && isEmailNotEmpty){
87.                 System.out.println("All textfield is not empty!");
88.                 resetAllStyle();
89.
90.                 // ตรวจสอบ Password ว่าตรงกันมั้ย
91.                 boolean isPasswordMatch = FieldValidation.isTextMatch(txtPassword,
    txtConfirmPassword, labelConfirmPassword, "ใส่รหัสผ่านไม่ตรงกัน");
92.                 if(isPasswordMatch){
93.                     CinemaUtility.resetTextStyle(txtPassword, labelPassword, "text-
    field-error");
94.                     CinemaUtility.resetTextStyle(txtConfirmPassword, labelConfirmPa
    ssword, "text-field-error");
95.                     System.out.println("Password is match.");
96.                 }
97.
98.                 // ดึงข้อมูลทั้งหมดใส่ List
99.                 uc.updateUserList();
100.                    allUser = uc.getUserList();
101.
102.                    // ค้นหา Username ซ้ำ
103.                    boolean searchUsernameFound = false;
104.                    searchUsernameFound = uc.checkExistUsername(txtUsername.getT
    ext());
105.
106.
107.                        // ถ้าเจอ Username แสดง error ว่ามีแล้ว
108.                        if(searchUsernameFound){
109.                            labelUsername.setText("ชื่อผู้ใช้งานนี้ถูกใช้งานแล้ว");
110.                            labelUsername.getStyleClass().add("text-error");
111.                            // ให้ Textfield เปลี่ยนเป็นสี Error
112.                            txtUsername.getStyleClass().add("text-field-
    error");
113.                        }else{
```

```
114.                                 CinemaUtility.resetTextStyle(txtUsername, labelUsern
    ame, "text-field-error");
115.                             }
116.
117.
118.                         // ตรวจสอบว่า email ถูกต้องมั้ย
119.                         boolean searchEmailFound = false;
120.                         boolean isEmailValid = FieldValidation.isValidEmailAddress(t
    xtEmail.getText(), txtEmail, labelEmail, "ใส่อีเมลไม่ถูกต้อง");
121.                         if(isEmailValid){
122.
123.                         CinemaUtility.resetTextStyle(txtEmail, labelEmail, "text-
    field-error");
124.                             System.out.println("Email is valid.");
125.
126.                             // ตรวจสอบว่าอีเมลซ้ำมั้ย
127.                             searchEmailFound = uc.checkExistEmail(txtEmail.getText()
    );
128.
129.                                 // ถ้าเจอ Email แสดง error
130.                                 if(searchEmailFound){
131.                                     labelEmail.setText("อีเมลนี้ถูกใช้งานแล้ว");
132.                                     labelEmail.getStyleClass().add("text-error");
133.                                     // ให้ Textfield เปลี่ยนเป็นสี Error
134.                                     txtEmail.getStyleClass().add("text-field-
    error"); // บอก Confirm password ไม่ตรง
135.                                 }else{
136.                                     CinemaUtility.resetTextStyle(txtEmail, labelEmai
    l, "text-field-error");
137.                                 }
138.                             }
139.
140.
141.                         // ถ้าทุกอย่างผ่านหมด Password ตรง , Username ไม่ซ้ำ , email ถูกต้องและไม่ซ้ำ
142.                         if(isPasswordMatch && !searchUsernameFound && isEmailValid &
    & !searchEmailFound){
143.                             System.out.println("Create Operation...");
144.                             // สร้าง Object
145.                             String username = txtUsername.getText();
146.                             String password = txtPassword.getText();
147.                             String firstname = txtFirstName.getText();
148.                             String lastname = txtLastName.getText();
149.                             String email = txtEmail.getText();
150.                             // 0 = admin , 1 = staff , 2 = customer
151.                             String type = "Customer"; // Default Customer
152.           //                  AlertMaker.showSimpleAlert("Password Prehash",password
    );
153.                             User newUser = new User(username, password, firstname, l
    astname, email, type, 0.0);
154.                             newUser.setEncryptPassword();
155.           //                  AlertMaker.showSimpleAlert("Password hashed",newUser.g
    etPassword());
156.           //                  AlertMaker.showSimpleAlert("Test pass", newUser.encryp
    tPassword(password));
157.           //                  boolean testPass = newUser.getPassword().equals(newUse
    r.encryptPassword(password));
158.           //                  String testPassStr = String.valueOf(testPass);
159.
160.       //                AlertMaker.showSimpleAlert("Euqal ?", testPassStr);
161.                             uc.addUser(newUser); // add new user
162.           //                  System.out.println("Add user to database cinema comple
    ted");
163.           //                  AlertMaker.showSimpleAlert("Register Completed", newUs
    er.toString());
```

```
164.            //                  boolean test = uc.checkValidUser(username, password);
165.            //                  String testStr = String.valueOf(test);
166.            //                  AlertMaker.showSimpleAlert("Test",testStr);
167.                                clearForm();
168.                                AlertMaker.showSimpleAlert("Register completed", newUser
    .toString());
169.
170.                                Parent parent;
171.                                parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/auth/Login.fxml"));
172.                                Scene parentScene = new Scene(parent);
173.                                Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
174.                                window.setScene(parentScene);
175.                                window.show();
176.
177.                            }
178.
179.                        }else{
180.                            if(isUernameNotEmpty){
181.                                CinemaUtility.resetTextStyle(txtUsername, labelUsername,
     "text-field-error");
182.                            }
183.
184.                            if(isPasswordNotEmpty){
185.                                CinemaUtility.resetTextStyle(txtPassword, labelPassword
    , "text-field-error");
186.                            }
187.
188.                            if(isConfirmPasswordNotEmpty){
189.                                CinemaUtility.resetTextStyle(txtConfirmPassword, labelCo
    nfirmPassword , "text-field-error");
190.                            }
191.
192.                            if(isFirstNameNotEmpty){
193.                                CinemaUtility.resetTextStyle(txtFirstName, labelFirstNam
    e , "text-field-error");
194.                            }
195.
196.
197.                        if(isLastNameNotEmpty){
198.                                CinemaUtility.resetTextStyle(txtLastName, labelLastName
    , "tex;t-field-error");
199.                            }
200.
201.                            if(isEmailNotEmpty){
202.                                CinemaUtility.resetTextStyle(txtEmail, labelEmail , "tex
    t-field-error");
203.                            }
204.
205.                        }
206.                    }
207.
208.
209.                    @FXML
210.                    private void handleClear(ActionEvent event) {
211.                        clearForm();
212.                        resetAllStyle();
213.                    }
214.
215.                    // ลบค่า Form
216.                    public void clearForm(){
217.                        txtUsername.setText("");
218.                        txtPassword.setText("");
219.                        txtConfirmPassword.setText("");
```

```
220.                     txtFirstName.setText("");
221.                     txtLastName.setText("");
222.                     txtEmail.setText("");
223.                 }
224.
225.                 public void clearLabel(){
226.                     labelUsername.setText("");
227.                     labelPassword.setText("");
228.                     labelConfirmPassword.setText("");
229.                     labelFirstName.setText("");
230.                     labelLastName.setText("");
231.                     labelEmail.setText("");
232.
233.                 }
234.
235.                 public void resetAllStyle(){
236.                     CinemaUtility.resetTextStyle(txtUsername, labelUsername, "text-
    field-error");
237.                     CinemaUtility.resetTextStyle(txtPassword, labelPassword , "text-
    field-error");
238.                     CinemaUtility.resetTextStyle(txtConfirmPassword, labelConfirmPas
    sword , "text-field-error");
239.                     CinemaUtility.resetTextStyle(txtFirstName, labelFirstName , "tex
    t-field-error");
240.                     CinemaUtility.resetTextStyle(txtLastName, labelLastName , "text-
    field-error");
241.                     CinemaUtility.resetTextStyle(txtEmail, labelEmail , "text-field-
    error");
242.                 }
243.
244.
245.                 @FXML
246.                 private void handleBack(ActionEvent event) {
247.                    try {
248.                         //((Stage)rootPane.getScene().getWindow()).close();
249.                         Parent parent;
250.                         parent = FXMLLoader.load(getClass().getResource("/cinema/ui/
    auth/Login.fxml"));
251.                         Scene parentScene = new Scene(parent);
252.                         Stage window = (Stage)((Node)event.getSource()).getScene().g
    etWindow();
253.                         window.setScene(parentScene);
254.                         window.show();
255.                     } catch (IOException ex) {
256.                         System.out.println(ex);
257.                     }
258.                 }
259.
260.             }
```

หน้าแสดงรายการภาพยนตร์

Class showmovieController

```
1.  package cinema.ui.showmovie;
2.
3.
4.      import cinema.CinemaController;
5.      import cinema.Movie;
6.      import cinema.UserController;
7.      import cinema.ui.AlertMaker;
8.      import cinema.ui.showDetailsMovie.MovieDetailController;
9.      import com.jfoenix.controls.JFXTabPane;
10.     import java.io.IOException;
11.     import java.io.InputStream;
12.     import java.net.URL;
13.     import java.time.LocalDate;
14.     import java.time.format.DateTimeFormatter;
15.     import java.time.temporal.ChronoUnit;
16.     import java.util.ArrayList;
17.     import java.util.List;
18.     import java.util.ResourceBundle;
19.     import javafx.fxml.FXML;
20.     import javafx.fxml.FXMLLoader;
21.     import javafx.fxml.Initializable;
22.     import javafx.geometry.Insets;
23.     import javafx.geometry.Pos;
24.     import javafx.scene.Node;
25.     import javafx.scene.Parent;
26.     import javafx.scene.Scene;
27.     import javafx.scene.control.Button;
28.     import javafx.scene.control.Label;
29.     import javafx.scene.control.ScrollPane;
30.     import javafx.scene.control.ScrollPane.ScrollBarPolicy;
31.     import javafx.scene.control.SingleSelectionModel;
32.     import javafx.scene.control.Tab;
33.     import javafx.scene.image.Image;
34.     import javafx.scene.image.ImageView;
35.     import javafx.scene.input.MouseEvent;
36.     import javafx.scene.layout.HBox;
37.     import javafx.scene.layout.StackPane;
38.     import javafx.scene.layout.VBox;
39.     import javafx.scene.paint.Color;
40.     import javafx.scene.text.Font;
41.     import javafx.scene.text.Text;
42.
43. import javafx.stage.Stage;
44.
45.
46.     /**
47.      * FXML Controller class
48.      *
49.      * @author BEAMCONAN
50.      */
51.     public class showmovieController implements Initializable {
52.
53.
54.         @FXML
55.         private ScrollPane scrollp1;
56.         @FXML
57.         private VBox field;
58.         @FXML
59.         private ScrollPane scrollp2;
```

```
60.         @FXML
61.         private VBox field2;
62.         @FXML
63.         private StackPane rootPane;
64.
65.
66.         CinemaController cc;
67.         UserController uc;
68.         List<Movie> movieList = new ArrayList<Movie>();
69.
70.
71.
72.         @FXML
73.         private Label labelAuthen;
74.         @FXML
75.         private JFXTabPane tabPane;
76.         @FXML
77.         private Label labelMyBooking;
78.
79.
80.         public showmovieController() {
81.             this.cc = cc.getInstance();
82.             this.uc = uc.getInstance();
83.         }
84.
85.
86.
87.
88.         @Override
89.         public void initialize(URL url, ResourceBundle rb) {
90.             if(uc.getIsLogin()){
91.                 labelAuthen.setText(uc.getLoginUser().getEmail() + " -> Logout");
92.                 labelMyBooking.setVisible(true);
93.             }else{
94.                 labelAuthen.setText("Login");
95.
96.             labelMyBooking.setVisible(false);
97.             }
98.
99.             SingleSelectionModel<Tab> selectionModel = tabPane.getSelectionModel();

100.                 if(cc.getIsComingSoon()){
101.                     selectionModel.select(1); // select theatre
102.                 }else{
103.                     selectionModel.select(0); // select theatre
104.                 }
105.
106.                 //************************************** Now showing ********
    ***************************************
107.                 movieList = cc.getMovieList();
108.                 /* ----------------------- check is now showing ----------------
    -------------- */
109.                 int showingDay = 14; // จำนวนวันที่จะให้หนังอยู่ในโรง
110.                 LocalDate today = LocalDate.now();
111.                 List<Movie> nowShowingList = new ArrayList<>();
112.                 List<Movie> comingSoonList = new ArrayList<>();
113.                 LocalDate releaseDate;
114.                 long tempDayShowing; // ฉายมาแล้วกี่วัน (Return เป็น long)
115.                 int dayShowing;// เปลี่ยน tempDayShowing เป็น int และไปบวกกับ showingDay
116.                 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMM
    M yyyy");
117.
118.                 for  (Movie movie : movieList) {
119.                     String date = movie.getReleaseDate();
120.                     releaseDate = LocalDate.parse(date, formatter);
```

```
121.
122.
123.                         tempDayShowing = today.until(releaseDate, ChronoUnit.DAYS);
    // ฉายมาแล้วกี่วัน ( จะได้ค่าติดลบจำนวนวัน )
124.                         dayShowing = (int) tempDayShowing + showingDay; // เหลือเวลาฉายอีกกี่
    วัน
125.                         if(today.until(releaseDate, ChronoUnit.DAYS) > 0) {
126.                             // เป็น coming soon
127.                             comingSoonList.add(movie);
128.                         }
129.                         else {
130.                             // เป็น now showing
131.                             if(dayShowing >= 0) {
132.                                 nowShowingList.add(movie);
133.                             }
134.                         }
135.                     }
136.
137.
138.
139.
140.                     int amountN = nowShowingList.size(); // count movie
141.                     Movie[] movieN = new Movie[amountN];
142.                     for (int i = 0; i < nowShowingList.size(); i++) {
143.                         movieN[i] = nowShowingList.get(i);
144.                     }
145.                     VBox[] vboxN = new VBox[amountN];
146.                     ImageView[] movieImgN = new ImageView[amountN];
147.                     Text[] nameEN = new Text[amountN];
148.                     Text[] nameTN = new Text[amountN];
149.                     Button[] btN = new Button[amountN];
150.                     Integer[] movieId = new Integer[amountN];
151.                     for (int i = 0; i < amountN; i++) {
152.                         InputStream image = getClass().getResourceAsStream("/cinema/
    posterImage/" + movieN[i].getPoster());
153.                         Image movie_image = new Image(image, 185, 275, false, false)
    ;
154.                         movieImgN[i] = new ImageView(movie_image);
155.                         String e = movieN[i].getEnglishName();
156.                         if (e.length() > 20) {
157.                             e = e.substring(0, 20) + "...";
158.                         }
159.                         nameEN[i] = new Text(185, 30, e);
160.                         nameEN[i].setFont(Font.font("Sukhumvit Set Semi Bold", 28));

161.                         nameEN[i].setFill(Color.web("0xFFFFFF"));
162.                         String t = movieN[i].getThaiName();
163.                         if (t.length() > 20) {
164.                             t = t.substring(0, 20) + "...";
165.                         }
166.                         nameTN[i] = new Text(185, 30, t);
167.                         nameTN[i].setFont(Font.font("Sukhumvit Set Text", 20));
168.                         nameTN[i].setFill(Color.web("0xFFFFFF"));
169.                         btN[i] = new Button("BUY NOW");
170.                         btN[i].setId("BUY NOW" + i);
171.                         //movieId[i] = movieN[i].getId();
172.                         int finalI = i;
173.                         btN[i].setOnAction(event -> {
174.                             try {
175.                                 //checkID(btN[finalI]);
176.                                 cc.setSelectMovie(nowShowingList.get(finalI).getId()
    );
177.                                 cc.setIsComingSoon(false);
178.           //                      System.out.println(cc.getSelectMovie().getId());
179.
```

```
180.                             Parent parent;
181.             //                      parent = FXMLLoader.load(getClass().getResource("/
    cinema/ui/showDetailsMovie/showMovieDetail.fxml"));
182.                                 if(uc.getIsLogin()){
183.                                     if(uc.getLoginUser().getType().equals("Staff")){

184.                                         parent = FXMLLoader.load(getClass().getResou
    rce("/cinema/ui/showtime/showtime.fxml"));
185.                                         Scene parentScene = new Scene(parent);
186.                                         Stage window = (Stage)((Node)event.getSource
    ()).getScene().getWindow();
187.                                         window.setScene(parentScene);
188.                                         window.show();
189.
190.
191.             //                            MovieDetailController.mediaPlayer.stop();

192.                                     }
193.                                     else {
194.                                         parent = FXMLLoader.load(getClass().getResou
    rce("/cinema/ui/showDetailsMovie/showMovieDetail.fxml"));
195.                                         Scene parentScene = new Scene(parent);
196.                                         Stage window = (Stage)((Node)event.getSource
    ()).getScene().getWindow();
197.                                         window.setScene(parentScene);
198.                                         window.show();
199.                                     }
200.                                 }
201.                                 else {
202.                                     parent = FXMLLoader.load(getClass().getResource(
    "/cinema/ui/showDetailsMovie/showMovieDetail.fxml"));
203.                                     Scene parentScene = new Scene(parent);
204.                                     Stage window = (Stage)((Node)event.getSource()).
    getScene().getWindow();
205.                                     window.setScene(parentScene);
206.                                     window.show();
207.                                 }
208.             //                      Scene parentScene = new Scene(parent);
209.             //                      Stage window = (Stage)((Node)event.getSource()).ge
    tScene().getWindow();
210.             //                      window.setScene(parentScene);
211.             //                      window.show();
212.                             } catch (IOException ex) {
213.
214.                             System.out.println(ex);
215.                             }
216.
217.
218.                         });
219.                         btN[i].setMaxWidth(180);
220.                         btN[i].getStyleClass().add("button-account");
221.                         btN[i].setFont(Font.font("Sukhumvit Set Semi Bold", 20));
222.
223.
224.                         vboxN[i] = new VBox(movieImgN[i], nameEN[i], nameTN[i], btN[
    i]);
225.                         vboxN[i].setPrefWidth(50);
226.                         vboxN[i].setAlignment(Pos.CENTER);
227.                         vboxN[i].setPrefSize(295, 300);
228.                         VBox.setMargin(nameEN[i], new Insets(5, 0, 0, 0));
229.                         VBox.setMargin(nameTN[i], new Insets(5, 0, 0, 0));
230.                         VBox.setMargin(btN[i], new Insets(10, 0, 0, 0));
231.                         HBox.setMargin(vboxN[i], new Insets(10, 10, 10, 10));
232.                     }
233.                     HBox movieRaw_hboxN[] = new HBox[4];
234.                     VBox movieLine_vboxN = new VBox();
```

```
235.                     int i = 0;
236.                     //System.out.print(Math.floor(amountN / 4));
237.                     while (i <= Math.floor(amountN / 4)) {
238.                         movieRaw_hboxN[i] = new HBox();
239.                         int j = 0;
240.                         while (j < 4 && (i * 4) + j < movieN.length) {
241.                             movieRaw_hboxN[i].getChildren().addAll(vboxN[(i * 4) + j
    ]);
242.                             j++;
243.                         }
244.                         movieLine_vboxN.getChildren().addAll(movieRaw_hboxN[i]);
245.                         i++;
246.                     }
247.                     movieLine_vboxN.setPrefSize(1280, movieLine_vboxN.getHeight());

248.                     scrollp1.setVbarPolicy(ScrollBarPolicy.AS_NEEDED);
249.                     field.getChildren().add(movieLine_vboxN);
250.                     scrollp1.setContent(field);
251.
252.                 //************************************** Coming Soon ***************
    ******************************
253.                     //movieList = cc.getMovieList();
254.                     int amountC = comingSoonList.size();
255.                     Movie[] movieC = new Movie[amountC];
256.                     for (int j = 0; j < comingSoonList.size(); j++) {
257.                         movieC[j] = comingSoonList.get(j);
258.
259.                 }
260.                     VBox[] vboxC = new VBox[amountC];
261.                     ImageView[] movieImgC = new ImageView[amountC];
262.                     Text[] nameEC = new Text[amountC];
263.                     Text[] nameTC = new Text[amountC];
264.                     Button[] btC = new Button[amountC];
265.                     for (i = 0; i < amountC; i++) {
266.                         InputStream image = getClass().getResourceAsStream("/cinema/
    posterImage/" + movieC[i].getPoster());
267.                         Image movie_image = new Image(image, 185, 275, false, false)
    ;
268.                         movieImgC[i] = new ImageView(movie_image);
269.                         String e = movieC[i].getEnglishName();
270.                         if (e.length() > 20) {
271.                             e = e.substring(0, 20) + "...";
272.                         }
273.                         nameEC[i] = new Text(185, 30, e);
274.                         nameEC[i].setFont(Font.font("Sukhumvit Set Semi Bold", 28));

275.                         nameEC[i].setFill(Color.web("0xFFFFFF"));
276.                         String t = movieC[i].getThaiName();
277.                         if (t.length() > 20) {
278.                             t = t.substring(0, 20) + "...";
279.                         }
280.                         nameTC[i] = new Text(185, 30, t);
281.                         nameTC[i].setFont(Font.font("Sukhumvit Set Text", 20));
282.                         nameTC[i].setFill(Color.web("0xFFFFFF"));
283.                         btC[i] = new Button("SELECT");
284.                         btC[i].setId("SELECT" + i);
285.                         int finalI = i;
286.                         btC[i].setOnAction(event -> {
287.                            try {
288.                                 //checkID(btN[finalI]);
289.                                 cc.setSelectMovie(comingSoonList.get(finalI).getId()
    );
290.                                 cc.setIsComingSoon(true);
291.                                 Parent parent;
292.                                 parent = FXMLLoader.load(getClass().getResource("/ci
    nema/ui/showDetailsMovie/showMovieDetail.fxml"));
```

```
293.            //                      if(uc.getIsLogin()){
294.            //                          if(uc.getLoginUser().getType().equals("Staff")
    ){
295.
296.        //                               parent = FXMLLoader.load(getClass().getResour
    ce("/cinema/ui/showtime/showtime.fxml"));
297.            //                          }
298.            //                      }
299.                                    Scene parentScene = new Scene(parent);
300.                                    Stage window = (Stage)((Node)event.getSource()).getS
    cene().getWindow();
301.                                    window.setScene(parentScene);
302.                                    window.show();
303.                                } catch (IOException ex) {
304.                                    System.out.println(ex);
305.                                }
306.                            });
307.                            btC[i].setMaxWidth(180);
308.                            btC[i].getStyleClass().add("button-account");
309.                            btC[i].setFont(Font.font("Sukhumvit Set Semi Bold", 20));
310.
311.
312.                            vboxC[i] = new VBox(movieImgC[i], nameEC[i], nameTC[i], btC[
    i]);
313.                            vboxC[i].setPrefWidth(50);
314.                            vboxC[i].setAlignment(Pos.CENTER);
315.                            vboxC[i].setPrefSize(295, 300);
316.                            VBox.setMargin(nameEC[i], new Insets(5, 0, 0, 0));
317.                            VBox.setMargin(nameTC[i], new Insets(5, 0, 0, 0));
318.                            VBox.setMargin(btC[i], new Insets(10, 0, 0, 0));
319.                            HBox.setMargin(vboxC[i], new Insets(10, 10, 10, 10));
320.                        }
321.                        HBox movieRaw_hboxC[] = new HBox[4];
322.                        VBox movieLine_vboxC = new VBox();
323.                        i = 0;
324.                        //System.out.print(Math.floor(amountC / 4));
325.                        while (i <= Math.floor(amountC / 4)) {
326.                            movieRaw_hboxC[i] = new HBox();
327.                            int j = 0;
328.                            while (j < 4 && (i * 4) + j < movieC.length) {
329.                                movieRaw_hboxC[i].getChildren().addAll(vboxC[(i * 4) + j
    ]);
330.                                j++;
331.                            }
332.                            movieLine_vboxC.getChildren().addAll(movieRaw_hboxC[i]);
333.                            i++;
334.                        }
335.                        movieLine_vboxC.setPrefSize(1280, movieLine_vboxC.getHeight());

336.                        scrollp2.setVbarPolicy(ScrollBarPolicy.AS_NEEDED);
337.                        field2.getChildren().add(movieLine_vboxC);
338.                        scrollp2.setContent(field2);
339.                }
340.
341.
342.                @FXML
343.                private void handleAuthen(MouseEvent event) throws IOException{
344.                    if (event.isPrimaryButtonDown() && event.getClickCount() == 1) {

345.                        if(uc.getIsLogin()){
346.                            uc.setIsLogin(false);
347.                            uc.unsetLoginUser();
348.                            cc.setIsComingSoon(false);
349.                        }
350.
351.                            Parent parent;
```

```
352.                             parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/auth/Login.fxml"));
353.                             Scene parentScene = new Scene(parent);
354.                             Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
355.                             window.setScene(parentScene);
356.                             window.show();
357.                         }
358.                         System.out.println("CLICK");
359.                 }
360.
361.
362.                 @FXML
363.                 private void handleBooking(MouseEvent event) throws IOException {
364.                     if (event.isPrimaryButtonDown() && event.getClickCount() == 1) {

365.                         if(uc.getIsLogin()){
366.                             Parent parent;
367.                             parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/myBooking/MyBooking.fxml"));
368.                             Scene parentScene = new Scene(parent);
369.                             Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
370.                             window.setScene(parentScene);
371.                             window.show();
372.                         }
373.
374.                     }
375.                 }
376.             }
```

หน้าดูรายละเอียดภาพยนตร์

Class MovieDetailController

```
1.  package cinema.ui.showDetailsMovie;
2.
3.  import cinema.CinemaController;
4.  import cinema.Movie;
5.  import cinema.Showtime;
6.  import cinema.UserController;
7.  import cinema.ui.AlertMaker;
8.  import java.io.IOException;
9.  import java.io.InputStream;
10. import javafx.fxml.FXML;
11. import javafx.fxml.Initializable;
12. import javafx.scene.image.ImageView;
13. import javafx.scene.text.Text;
14.
15. import java.net.URL;
16. import java.util.List;
17. import java.util.ResourceBundle;
18. import java.util.logging.Level;
19. import java.util.logging.Logger;
20. import javafx.event.ActionEvent;
21. import javafx.fxml.FXMLLoader;
22. import javafx.scene.Node;
23. import javafx.scene.Parent;
24. import javafx.scene.Scene;
25. import javafx.scene.control.Button;
26. import javafx.scene.image.Image;
27. import javafx.scene.input.MouseEvent;
28. import javafx.scene.layout.VBox;
29. import javafx.scene.media.Media;
30. import javafx.scene.media.MediaPlayer;
31. import javafx.scene.media.MediaView;
32. import javafx.stage.Stage;
33. import javafx.util.Duration;
34.
35. public class MovieDetailController implements Initializable {
36.
37.     @FXML
38.     private ImageView movieImg;
39.
40.     @FXML
41.     private Text movieNameEng;
42.
43.     @FXML
44.     private Text movieNameTh;
45.
46.     @FXML
47.     private Text releaseDate;
48.
49.     @FXML
50.     private Text genre;
51.
52.     @FXML
53.     private Text actor;
54.
55.     @FXML
56.     private Text synopsis;
57.     @FXML
58.     private MediaView mediaV;
59.     @FXML
```

```
60.     private Text time;
61.     @FXML
62.     private Text director;
63.
64.     public MediaPlayer mediaPlayer;
65.     boolean play = true;
66.
67.     Movie movie;
68.     CinemaController cc;
69.     @FXML
70.     private VBox bottomButton;
71.     @FXML
72.     private Button btnNext;
73.
74.     UserController uc;
75.     public MovieDetailController() {
76.         this.cc = cc.getInstance();
77.         this.uc = uc.getInstance();
78.     }
79.
80.     @Override
81.     public void initialize(URL location, ResourceBundle resources) {
82.         System.out.println(cc.getIsComingSoon());
83.         if(cc.getIsComingSoon()){
84.             btnNext.setVisible(false);
85.         }
86.
87.
88.         // ใส่ ID เข้าไป
89.         movie = cc.getSelectMovie();
90. //        System.out.println(movie);
91.         InputStream imagePoster = getClass().getResourceAsStream("/cinema/posterIma
    ge/" + movie.getPoster());
92. //        URL videoPoster = getClass().getResource("/cinema/ui/video/"+movie.getTra
    iler());
93.
94.         Image movie_image = new Image(imagePoster, 223, 301, false, false);
95.         movieImg.setImage(movie_image);
96.         movieNameEng.setText(movie.getEnglishName());
97.         //movieNameEng = new Text(movie.getEnglishName());
98.         movieNameTh.setText(movie.getThaiName());
99.         releaseDate.setText(movie.getReleaseDate());
100.                time.setText(movie.getTimeMinute());
101.                genre.setText(movie.getGenre());
102.                director.setText(movie.getDirector());
103.                actor.setText(movie.getCast());
104.                synopsis.setText(movie.getSynopsis());
105.
106.                Media media = new Media(movie.getTrailer());
107.       //
108.                mediaPlayer = new MediaPlayer(media);
109.                mediaPlayer.setAutoPlay(true);
110.                mediaPlayer.play();
111.                mediaV.setMediaPlayer(mediaPlayer);
112.       //****************** webView**********************
113.       //        webV.getEngine().load("https://youtu.be/TiblmGnet2Q");
114.           }
115.
116.           @FXML
117.           private void nextPage(ActionEvent event) throws IOException {
118.               mediaPlayer.stop();
119.               mediaPlayer.seek(Duration.ZERO);
120.
121.               Parent parent;
122.               parent = FXMLLoader.load(getClass().getResource("/cinema/ui/showtime
    /showtime.fxml"));
```

```
123.                Scene parentScene = new Scene(parent);
124.                Stage window = (Stage)((Node)event.getSource()).getScene().getWindow
    ();
125.                window.setScene(parentScene);
126.                window.show();
127.            }
128.
129.            @FXML
130.            private void back(ActionEvent event) throws IOException {
131.
132.                mediaPlayer.stop();
133.                mediaPlayer.seek(Duration.ZERO);
134.
135.                Parent parent;
136.                            parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/showmovie/showmovie.fxml"));
137.                            Scene parentScene = new Scene(parent);
138.                            Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
139.                            window.setScene(parentScene);
140.                            window.show();
141.            }
142.
143.            @FXML
144.            private void clickVideo(MouseEvent event) {
145.                boolean checkEndOfTrailer = mediaPlayer.getCurrentTime().equals(medi
    aPlayer.getStopTime());
146.                // Trailer is not end.
147.                if (!checkEndOfTrailer) {
148.                    if (play) {
149.                        mediaPlayer.pause();
150.                        play = false;
151.                    } else {
152.                        mediaPlayer.play();
153.                        play = true;
154.                    }
155.                } // Trailer is end.
156.                else {
157.                    mediaPlayer.seek(Duration.ZERO);
158.                    mediaPlayer.play();
159.                    play = true;
160.                }
161.            }
162.
163.
164.        }
```

หน้าเลือกรอบฉายภาพยนตร์

Class ShowtimeController

```
1.  package cinema.ui.showtime;
2.
3.
4.      import cinema.CinemaController;
5.      import cinema.Showtime;
6.      import cinema.Theatre;
7.      import cinema.UserController;
8.      import cinema.ui.AlertMaker;
9.      import cinema.ui.showDetailsMovie.MovieDetailController;
10.     import com.jfoenix.controls.JFXDatePicker;
11.     import java.io.IOException;
12.     import java.net.URL;
13.     import java.time.LocalDate;
14.     import java.time.format.DateTimeFormatter;
15.     import java.time.temporal.ChronoUnit;
16.     import java.util.ArrayList;
17.     import java.util.List;
18.     import java.util.ResourceBundle;
19.     import javafx.event.ActionEvent;
20.     import javafx.fxml.FXML;
21.     import javafx.fxml.FXMLLoader;
22.     import javafx.fxml.Initializable;
23.     import javafx.geometry.Insets;
24.     import javafx.geometry.Pos;
25.     import javafx.scene.Node;
26.     import javafx.scene.Parent;
27.     import javafx.scene.Scene;
28.     import javafx.scene.control.Button;
29.     import javafx.scene.control.ScrollPane;
30.     import javafx.scene.layout.HBox;
31.     import javafx.scene.layout.StackPane;
32.     import javafx.scene.layout.VBox;
33.     import javafx.scene.paint.Color;
34.     import javafx.scene.text.Font;
35.     import javafx.scene.text.Text;
36.     import javafx.stage.Stage;
37.
38.
39.     public class ShowtimeController implements Initializable{
40.
41.
42.         @FXML
43.         private Text nameEng;
44.
45.     @FXML
46.         private JFXDatePicker date;
47.         @FXML
48.         private VBox vbox;
49.         @FXML
50.         private ScrollPane scroll;
51.         //Movie movie;
52.         CinemaController cc;
53.         UserController uc;
54.         @FXML
55.         private StackPane rootPane;
56.         @FXML
57.         private Text textNotHaveShowtime;
58.         HBox hbox_showtimepertheatre = new HBox();
59.
```

```
60.         public ShowtimeController() {
61.             this.cc = cc.getInstance();
62.             this.uc = uc.getInstance();
63.         }
64.
65.
66.         @Override
67.         public void initialize(URL url, ResourceBundle rb) {
68.             // TODO
69.
70.             date.setValue(LocalDate.now());
71.
72.             //hbox_showtimepertheatre.setVisible(true);
73.     //         hbox_showtimepertheatre.getChildren().removeAll();
74.             loadShowtime(date.getValue());
75.
76.         }
77.
78.         public void loadShowtime(LocalDate date) {
79.             String formattedString = null;
80.             try{
81.                 LocalDate ld = date;
82.                 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMMM y
    yyy");
83.                 formattedString = ld.format(formatter);
84.             }
85.             catch(Exception e) {
86.                 System.out.println(e);
87.             }
88.
89.             List<Theatre> theatreList = cc.getTheatreList();
90.             /* --------- ได้ showtime ที่มีหนังที่เรากดเข้ามา -----------*/
91.
92.         List<Showtime> showtimeHaveSelectedMovie = new ArrayList<>();
93.             List<Theatre> tList = cc.getTheatreList();
94.             boolean checkShowtimeExist = false;
95.             // เช็คว่า showtime มีหนังนั้นไหม ถ้ามีเพิ่มเข้า List
96.             for (Theatre t : theatreList) {
97.                 List <Showtime> stList = t.getShowtimeList();
98.                 for (Showtime s : stList) {
99.                     if(s.getDate().equals(formattedString)) {
100.                                if(s.getMovie().getId() == cc.getSelectMovie().getId
    ()){
101.                                    showtimeHaveSelectedMovie.add(s);
102.                                    checkShowtimeExist = true;
103.                                }
104.                            }
105.                    }
106.                }
107.
108.                nameEng.setText(cc.getSelectMovie().getEnglishName());
109.
110.                if(!checkShowtimeExist) {
111.                    textNotHaveShowtime.setText("ไม่มีรอบฉายในวันที่ที่ท่านเลือก");
112.                    //hbox_showtimepertheatre.setVisible(false);
113.                }else{
114.                    //hbox_showtimepertheatre.setVisible(true);
115.                    textNotHaveShowtime.setText(null);
116.
117.                    int amount = showtimeHaveSelectedMovie.size();
118.                    Integer[] showtimeId = new Integer[amount];
119.
120.
121.                    Showtime[] showtime = new Showtime[amount];
```

```
122.                         for (int i = 0; i < showtimeHaveSelectedMovie.size(); i++) {
123.                             showtime[i] = showtimeHaveSelectedMovie.get(i);
124.                         }
125.
126.
127.                         int i = 0, tempTheatre = showtime[0].getTheatre().getTheatre
    Number(), countT = 0;
128.                         Button[] bt = new Button[amount];
129.                         VBox vbox_theatre;
130.
131.
132.                         Text theatreNo = new Text(Integer.toString(showtime[i].getTh
    eatre().getTheatreNumber()));
133.
134.
135.                         theatreNo.setFont(Font.font("Sukhumvit Set Text", 22));
136.                         theatreNo.setFill(Color.web("0xFFFFFF"));
137.
138.                     Text detailsTheatre = new Text(showtime[i].getSystem() + " - " +
     showtime[i].getSoundtrack().toUpperCase() + "/" + showtime[i].getSubtitle().toUppe
    rCase());
139.                         detailsTheatre.setFont(Font.font("Sukhumvit Set Text", 18));
140.                         detailsTheatre.setFill(Color.web("0xFFFFFF"));
141.
142.
143.                         vbox_theatre = new VBox(theatreNo, detailsTheatre);
144.                         vbox_theatre.setPrefWidth(100);
145.                         vbox_theatre.setAlignment(Pos.CENTER);
146.                         hbox_showtimepertheatre = new HBox(vbox_theatre);
147.                         HBox.setMargin(vbox_theatre, new Insets(0, 20, 0, 10));
148.
149.
150.                         while (i < amount) {
151.                             if (tempTheatre != showtime[i].getTheatreNo()) {
152.                                 System.out.println("Noo: " + tempTheatre);
153.                                 if (countT != 0) {
154.                                     countT = 0;//0;
155.                                     tempTheatre = showtime[i].getTheatreNo();
156.                                     System.out.println("No: " + tempTheatre);
157.                                 }
158.                                 if (countT == 0) {
159.                                     vbox.getChildren().add(hbox_showtimepertheatre);
160.                                     theatreNo = new Text(Integer.toString(showtime[i
    ].getTheatreNo()));
161.                                     theatreNo.setFont(Font.font("Sukhumvit Set Text"
    , 22));
162.                                     theatreNo.setFill(Color.web("0xFFFFFF"));
163.                                     detailsTheatre = new Text(showtime[i].getSystem(
    ) + " - " + showtime[i].getSoundtrack().toUpperCase() + "/" + showtime[i].getSubtit
    le().toUpperCase());
164.                                     detailsTheatre.setFont(Font.font("Sukhumvit Set
    Text", 18));
165.                                     detailsTheatre.setFill(Color.web("0xFFFFFF"));
166.
167.
168.                                     vbox_theatre = new VBox(theatreNo, detailsTheatr
    e);
169.                                     vbox_theatre.setPrefWidth(100);
170.                                     vbox_theatre.setAlignment(Pos.CENTER);
171.                                     hbox_showtimepertheatre = new HBox(vbox_theatre)
    ;
172.                                     HBox.setMargin(vbox_theatre, new Insets(0, 20, 0
    , 10));
```

```
173.                                     bt[i] = new Button(showtime[i].getShowtime());
174.
175.                                 bt[i].setFont(Font.font("Sukhumvit Set Semi Text", 2
    2));
176.                                     bt[i].setMaxWidth(200);
177.                                     bt[i].getStyleClass().add("button-st");
178.                                     HBox.setMargin(bt[i], new Insets(5, 10, 5, 10));

179.                                     hbox_showtimepertheatre.getChildren().add(bt[i])
    ;
180.                                     showtimeId[i] = showtime[i].getId();
181.                                     System.err.println(showtimeId[i]);
182.                                     int finalI = i;
183.                                     bt[i].setOnAction(event -> {
184.                                         try {
185.                                             cc.setSelectShowtime(showtimeId[finalI])
    ;
186.
187.
188.                                             Parent parent;
189.                                             parent = FXMLLoader.load(getClass().getR
    esource("/cinema/ui/layoutMedium/layoutMedium.fxml"));
190.                                             Scene parentScene = new Scene(parent);
191.                                             Stage window = (Stage)((Node)event.getSo
    urce()).getScene().getWindow();
192.                                             window.setScene(parentScene);
193.                                             window.show();
194.                                         } catch (IOException ex) {
195.                                             System.out.println(ex);
196.                                         }
197.
198.
199.                                     });
200.                                     if (i == amount - 1) {
201.                                         vbox.getChildren().add(hbox_showtimepertheat
    re);
202.                                     }
203.                                 }
204.                             } else {
205.
206.
207.                                 bt[i] = new Button(showtime[i].getShowtime());
208.                                 bt[i].setFont(Font.font("Sukhumvit Set Semi Text", 2
    2));
209.                                 bt[i].setMaxWidth(200);
210.                                 bt[i].getStyleClass().add("button-st");
211.                                 HBox.setMargin(bt[i], new Insets(5, 10, 5, 10));
212.                                 hbox_showtimepertheatre.getChildren().add(bt[i]);
213.                                 showtimeId[i] = showtime[i].getId();
214.                                 System.err.println(showtimeId[i]);
215.
216.                             int finalI = i;
217.                                 bt[i].setOnAction(event -> {
218.                                     try {
219.                                             cc.setSelectShowtime(showtimeId[finalI])
    ;
220.
221.
222.                                             Parent parent;
223.                                             parent = FXMLLoader.load(getClass().getR
    esource("/cinema/ui/layoutMedium/layoutMedium.fxml"));
224.                                             Scene parentScene = new Scene(parent);
225.                                             Stage window = (Stage)((Node)event.getSo
    urce()).getScene().getWindow();
226.                                             window.setScene(parentScene);
227.                                             window.show();
```

```
                                    } catch (IOException ex) {
                                        System.out.println(ex);
                                    }


                                });
                                if (i == amount - 1) {
                                    vbox.getChildren().add(hbox_showtimepertheatre);

                                }
                            }
                            countT += 1;
                            i += 1;
                        }
                    }
                }


                @FXML
                private void getDate(ActionEvent event) {
                    System.out.println(date.getValue());
                    LocalDate today = LocalDate.now();
                    vbox.getChildren().clear();
                    if((today.until(date.getValue(), ChronoUnit.DAYS) < 0)) {
                        textNotHaveShowtime.setText("ไม่สามารถแสดงเวลาฉายที่น้อยกว่าวันที่ปัจจุบันได้");
                        //hbox_showtimepertheatre.setVisible(false);
            //            hbox_showtimepertheatre.getChildren().clear();
                    }
                    else {
                        if((today.until(date.getValue(), ChronoUnit.DAYS) > 7)) {
                            textNotHaveShowtime.setText("ไม่สามารถแสดงรายการเวลาฉายได้
    เกิน 7 วัน");
                            //hbox_showtimepertheatre.setVisible(false);

        //                  hbox_showtimepertheatre.getChildren().clear();
                        }
                        else {
                            //hbox_showtimepertheatre.setVisible(true);
            //                hbox_showtimepertheatre.getChildren().clear();
                            loadShowtime(date.getValue());
                        }
                    }
            //          hbox_showtimepertheatre.getChildren().clear();
                }


                @FXML
                private void back(ActionEvent event) throws IOException {
                    Parent parent;

                    if(uc.getIsLogin()){
                        if(uc.getLoginUser().getType().equals("Staff")){
                            parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/showmovie/showmovie.fxml"));
                            Scene parentScene = new Scene(parent);
                            Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
                            window.setScene(parentScene);
                            window.show();
                        }
                        else {
                            parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/showDetailsMovie/showMovieDetail.fxml"));
                            Scene parentScene = new Scene(parent);
```

```
                            Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
                            window.setScene(parentScene);
                            window.show();
                        }
                    }
                    else {
                        parent = FXMLLoader.load(getClass().getResource("/cinema/ui/
    showDetailsMovie/showMovieDetail.fxml"));
                        Scene parentScene = new Scene(parent);
                        Stage window = (Stage)((Node)event.getSource()).getScene().g
    etWindow();
                        window.setScene(parentScene);
                        window.show();
                    }
                }


            }
```

หน้าเลือกที่นั่งภายในโรงภาพยนตร์

Class Controller

```
1.  package cinema.ui.layoutMedium;
2.
3.
4.      import cinema.CinemaController;
5.      import cinema.Seat;
6.      import cinema.Showtime;
7.      import cinema.UserController;
8.      import cinema.ui.AlertMaker;
9.      import java.io.FileNotFoundException;
10.     import java.io.IOException;
11.     import java.io.InputStream;
12.     import java.net.URL;
13.     import java.util.ArrayList;
14.     import java.util.List;
15.     import java.util.Optional;
16.     import java.util.ResourceBundle;
17.     import javafx.event.ActionEvent;
18.     import javafx.fxml.FXML;
19.     import javafx.fxml.FXMLLoader;
20.     import javafx.fxml.Initializable;
21.     import javafx.scene.Node;
22.     import javafx.scene.Parent;
23.     import javafx.scene.Scene;
24.     import javafx.scene.control.Alert;
25.     import javafx.scene.control.Button;
26.     import javafx.scene.control.ButtonType;
27.     import javafx.scene.image.Image;
28.     import javafx.scene.image.ImageView;
29.     import javafx.scene.layout.AnchorPane;
30.     import javafx.scene.text.Text;
31.     import javafx.stage.Stage;
32.
33.
34.     /**
35.      *
36.      * @author BEAMCONAN
37.      */
38.     public class Controller implements Initializable {
39.
40.
41.         @FXML
42.         private ImageView F1;
43.         @FXML
44.
45.     private ImageView F2;
46.         @FXML
47.         private ImageView F3;
48.         @FXML
49.         private ImageView F4;
50.         @FXML
51.         private ImageView F6;
52.         @FXML
53.         private ImageView F7;
54.         @FXML
55.         private ImageView F8;
56.         @FXML
57.         private ImageView F9;
58.         @FXML
59.         private ImageView F10;
```

```
60.         @FXML
61.         private ImageView F11;
62.         @FXML
63.         private ImageView F12;
64.         @FXML
65.         private ImageView F13;
66.         @FXML
67.         private ImageView F14;
68.         @FXML
69.         private ImageView F15;
70.         @FXML
71.         private ImageView F16;
72.         @FXML
73.         private ImageView F17;
74.         @FXML
75.         private ImageView F18;
76.         @FXML
77.         private ImageView F19;
78.         @FXML
79.         private ImageView F20;
80.         @FXML
81.         private ImageView E1;
82.         @FXML
83.         private ImageView E2;
84.         @FXML
85.         private ImageView E3;
86.         @FXML
87.         private ImageView E4;
88.         @FXML
89.         private ImageView E5;
90.         @FXML
91.
92.     private ImageView E6;
93.         @FXML
94.         private ImageView E7;
95.         @FXML
96.         private ImageView E8;
97.         @FXML
98.         private ImageView E9;
99.         @FXML
100.                private ImageView E10;
101.                @FXML
102.                private ImageView E11;
103.                @FXML
104.                private ImageView E12;
105.                @FXML
106.                private ImageView E13;
107.                @FXML
108.                private ImageView E14;
109.                @FXML
110.                private ImageView E15;
111.                @FXML
112.                private ImageView E16;
113.                @FXML
114.                private ImageView E17;
115.                @FXML
116.                private ImageView E18;
117.                @FXML
118.                private ImageView E19;
119.                @FXML
120.                private ImageView E20;
121.                @FXML
122.                private ImageView D1;
123.                @FXML
124.                private ImageView D2;
125.                @FXML
```

```
126.                private ImageView D3;
127.                @FXML
128.                private ImageView D4;
129.                @FXML
130.                private ImageView D5;
131.                @FXML
132.                private ImageView D6;
133.                @FXML
134.                private ImageView D7;
135.                @FXML
136.                private ImageView D8;
137.                @FXML
138.
139.            private ImageView D9;
140.                @FXML
141.                private ImageView D10;
142.                @FXML
143.                private ImageView D11;
144.                @FXML
145.                private ImageView D12;
146.                @FXML
147.                private ImageView D13;
148.                @FXML
149.                private ImageView D14;
150.                @FXML
151.                private ImageView D15;
152.                @FXML
153.                private ImageView D16;
154.                @FXML
155.                private ImageView D17;
156.                @FXML
157.                private ImageView D18;
158.                @FXML
159.                private ImageView D19;
160.                @FXML
161.                private ImageView D20;
162.                @FXML
163.                private ImageView C1;
164.                @FXML
165.                private ImageView C2;
166.                @FXML
167.                private ImageView C3;
168.                @FXML
169.                private ImageView C4;
170.                @FXML
171.                private ImageView C5;
172.                @FXML
173.                private ImageView C6;
174.                @FXML
175.                private ImageView C7;
176.                @FXML
177.                private ImageView C8;
178.                @FXML
179.                private ImageView C9;
180.                @FXML
181.                private ImageView C10;
182.                @FXML
183.                private ImageView C11;
184.                @FXML
185.
186.            private ImageView C12;
187.                @FXML
188.                private ImageView C13;
189.                @FXML
190.                private ImageView C14;
191.                @FXML
```

```
192.                private ImageView C15;
193.                @FXML
194.                private ImageView C16;
195.                @FXML
196.                private ImageView C17;
197.                @FXML
198.                private ImageView C18;
199.                @FXML
200.                private ImageView C19;
201.                @FXML
202.                private ImageView C20;
203.                @FXML
204.                private ImageView B1;
205.                @FXML
206.                private ImageView B2;
207.                @FXML
208.                private ImageView B3;
209.                @FXML
210.                private ImageView B4;
211.                @FXML
212.                private ImageView B5;
213.                @FXML
214.                private ImageView B6;
215.                @FXML
216.                private ImageView B7;
217.                @FXML
218.                private ImageView B8;
219.                @FXML
220.                private ImageView B9;
221.                @FXML
222.                private ImageView B10;
223.                @FXML
224.                private ImageView B11;
225.                @FXML
226.                private ImageView B12;
227.                @FXML
228.                private ImageView B13;
229.                @FXML
230.                private ImageView B14;
231.                @FXML
232.
233.            private ImageView B15;
234.                @FXML
235.                private ImageView B16;
236.                @FXML
237.                private ImageView B17;
238.                @FXML
239.                private ImageView B18;
240.                @FXML
241.                private ImageView B19;
242.                @FXML
243.                private ImageView B20;
244.                @FXML
245.                private ImageView A3;
246.                @FXML
247.                private ImageView A4;
248.                @FXML
249.                private ImageView A5;
250.                @FXML
251.                private ImageView A2;
252.                @FXML
253.                private ImageView A1;
254.                @FXML
255.                private AnchorPane rootPane;
256.                @FXML
257.                private ImageView F5;
```

```
258.
259.                int count;
260.                boolean isFound;
261.                String maxIV;
262.                String minIV;
263.                List<String> ivList = new ArrayList<String>();
264.
265.                List<ImageView> ivListAll = new ArrayList<>(); // ใช้
    เก็บ Image View เพื่อ Disable
266.                List<String> ivListDisable = new ArrayList<>(); // ใช้เก็บ List ที่นั่งที่จองไปแล้ว
    ในรอบฉายนั้น
267.
268.                CinemaController cc;
269.                UserController uc;
270.                @FXML
271.                private Text txtShowtime;
272.                @FXML
273.                private Text labelPreSummary;
274.                @FXML
275.                private Text txtMovie;
276.                @FXML
277.
278.            private Button btnNext;
279.
280.                public Controller() {
281.                    this.cc = cc.getInstance();
282.                    this.uc = uc.getInstance();
283.                }
284.
285.
286.                @Override
287.                public void initialize(URL url, ResourceBundle rb) {
288.                    if(uc.getIsLogin()){
289.                        if(ivList.size() > 0){
290.                            btnNext.setDisable(false);
291.                        }else{
292.                            btnNext.setDisable(true);
293.                        }
294.                    }else{
295.                        btnNext.setDisable(true);
296.                    }
297.
298.
299.
300.                    labelPreSummary.setText(null);
301.                    isFound = false;
302.                    count = 0;
303.                    seatAssign();
304.                    // Get Showtime seat which is not available
305.                    System.out.println(cc.getSelectShowtime());
306.                    Showtime st = cc.getShowtime(cc.getSelectShowtime());
307.                    txtShowtime.setText("ผังที่นั่ง รอบฉาย : " + st.getShowtime());
308.                    txtMovie.setText(st.getShowtimeDetail());
309.
310.                    List<Seat> seatAlreadyBooked = st.getSeatList();
311.                    for (Seat seat : seatAlreadyBooked) {
312.                        if(seat.getSeatStatus()){ // if seat status is true => alrea
    dy book => disable it
313.                            ivListDisable.add(seat.getSeatName());
314.                        }
315.                    }
316.                    seatDisable(); // find in ivlist to disable seat
317.                }
318.
319.                public void seatAssign(){
```

```
320.                    ivListAll.add(A1);
321.                    ivListAll.add(A2);
322.                    ivListAll.add(A3);
323.                    ivListAll.add(A4);
324.
325.                ivListAll.add(A5);
326.                    ivListAll.add(B1);
327.                    ivListAll.add(B2);
328.                    ivListAll.add(B3);
329.                    ivListAll.add(B4);
330.                    ivListAll.add(B5);
331.                    ivListAll.add(B6);
332.                    ivListAll.add(B7);
333.                    ivListAll.add(B8);
334.                    ivListAll.add(B9);
335.                    ivListAll.add(B10);
336.                    ivListAll.add(B11);
337.                    ivListAll.add(B12);
338.                    ivListAll.add(B13);
339.                    ivListAll.add(B14);
340.                    ivListAll.add(B15);
341.                    ivListAll.add(B16);
342.                    ivListAll.add(B17);
343.                    ivListAll.add(B18);
344.                    ivListAll.add(B19);
345.                    ivListAll.add(B20);
346.                    ivListAll.add(C1);
347.                    ivListAll.add(C2);
348.                    ivListAll.add(C3);
349.                    ivListAll.add(C4);
350.                    ivListAll.add(C5);
351.                    ivListAll.add(C6);
352.                    ivListAll.add(C7);
353.                    ivListAll.add(C8);
354.                    ivListAll.add(C9);
355.                    ivListAll.add(C10);
356.                    ivListAll.add(C11);
357.                    ivListAll.add(C12);
358.                    ivListAll.add(C13);
359.                    ivListAll.add(C14);
360.                    ivListAll.add(C15);
361.                    ivListAll.add(C16);
362.                    ivListAll.add(C17);
363.                    ivListAll.add(C18);
364.                    ivListAll.add(C19);
365.                    ivListAll.add(C20);
366.                    ivListAll.add(D1);
367.                    ivListAll.add(D2);
368.                    ivListAll.add(D3);
369.                    ivListAll.add(D4);
370.                    ivListAll.add(D5);
371.
372.                ivListAll.add(D6);
373.                    ivListAll.add(D7);
374.                    ivListAll. add(D8);
375.                    ivListAll.add(D9);
376.                    ivListAll.add(D10);
377.                    ivListAll.add(D11);
378.                    ivListAll.add(D12);
379.                    ivListAll.add(D13);
380.                    ivListAll.add(D14);
381.                    ivListAll.add(D15);
382.                    ivListAll.add(D16);
383.                    ivListAll.add(D17);
384.                    ivListAll.add(D18);
385.                    ivListAll.add(D19);
```

```
386.                    ivListAll.add(D20);
387.                    ivListAll.add(E1);
388.                    ivListAll.add(E2);
389.                    ivListAll.add(E3);
390.                    ivListAll.add(E4);
391.                    ivListAll.add(E5);
392.                    ivListAll.add(E6);
393.                    ivListAll.add(E7);
394.                    ivListAll.add(E8);
395.                    ivListAll.add(E9);
396.                    ivListAll.add(E10);
397.                    ivListAll.add(E11);
398.                    ivListAll.add(E12);
399.                    ivListAll.add(E13);
400.                    ivListAll.add(E14);
401.                    ivListAll.add(E15);
402.                    ivListAll.add(E16);
403.                    ivListAll.add(E17);
404.                    ivListAll.add(E18);
405.                    ivListAll.add(E19);
406.                    ivListAll.add(E20);
407.                    ivListAll.add(F1);
408.                    ivListAll.add(F2);
409.                    ivListAll.add(F3);
410.                    ivListAll.add(F4);
411.                    ivListAll.add(F5);
412.                    ivListAll.add(F6);
413.                    ivListAll.add(F7);
414.                    ivListAll.add(F8);
415.                    ivListAll.add(F9);
416.                    ivListAll.add(F10);
417.                    ivListAll.add(F11);
418.
419.                ivListAll.add(F12);
420.                    ivListAll.add(F13);
421.                    ivListAll.add(F14);
422.                    ivListAll.add(F15);
423.                    ivListAll.add(F16);
424.                    ivListAll.add(F17);
425.                    ivListAll.add(F18);
426.                    ivListAll.add(F19);
427.                    ivListAll.add(F20);
428.
429.                    // Test for Disable all
430.            //        for (int i = 1; i <= 5; i++) { ivListDisable.add("A"+i); }
431.            //        for (int i = 1; i <= 20; i++) { ivListDisable.add("B"+i); }
432.            //        for (int i = 1; i <= 20; i++) { ivListDisable.add("C"+i); }
433.            //        for (int i = 1; i <= 20; i++) { ivListDisable.add("D"+i); }
434.            //        for (int i = 1; i <= 20; i++) { ivListDisable.add("E"+i); }
435.            //        for (int i = 1; i <= 20; i++) { ivListDisable.add("F"+i); }
436.
437.
438.            //        ivListDisable.add("A1");
439.            //        ivListDisable.add("A5");
440.            //        ivListDisable.add("B17");
441.
442.
443.                }
444.                public void seatDisable(){
445.                    for (ImageView iv : ivListAll) {
446.                        for (String id : ivListDisable) {
447.                            if(id.equals(iv.getId())){
448.                                //System.out.println("Found" + iv.getId());
449.                                iv.setDisable(true);
450.                                if(iv.getId().substring(0,1).equals("A")){
451.                                    // disable image for a
```

```
452.                                 InputStream imagePaired = getClass().getResource
    AsStream("/cinema/ui/images/pair1.png");
453.                                 iv.setImage(new Image(imagePaired));
454.                             }else if(iv.getId().substring(0,1).equals("B") || iv
    .getId().substring(0,1).equals("C")){
455.                                 // disable image for b
456.                                 InputStream imageHoneymoon = getClass().getResou
    rceAsStream("/cinema/ui/images/blue1.png");
457.
458.                             iv.setImage(new Image(imageHoneymoon));
459.                             }else{
460.                                 // disable for d-f
461.                                 InputStream imageNormal = getClass().getResource
    AsStream("/cinema/ui/images/red1.png");
462.                                 iv.setImage(new Image(imageNormal));
463.                             }
464.                         }
465.                     }
466.                 }
467.             }
468.
469.
470.             @FXML
471.             private void nextPage(ActionEvent event) throws IOException {
472.                 System.err.println("Booking Operation");
473.                 // Set value for booking
474.                 cc.setSeatList(ivList);
475.
476.                 Parent parent;
477.         //         parent = FXMLLoader.load(getClass().getResource("/cinema/ui/su
    mmary/summary.fxml"));
478.                 if(uc.getIsLogin()){
479.                     if(uc.getLoginUser().getType().equals("Staff")){
480.                         parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/summaryStaff/summaryStaff.fxml"));
481.                         Scene parentScene = new Scene(parent);
482.                         Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
483.                         window.setScene(parentScene);
484.                         window.show();
485.                     }
486.                     else {
487.                         parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/summary/summary.fxml"));
488.                         Scene parentScene = new Scene(parent);
489.                         Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
490.                         window.setScene(parentScene);
491.                         window.show();
492.                     }
493.                 }
494.                 else {
495.                     Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
496.
497.                 alert.setTitle("Login Required");
498.                     alert.setContentText("Please login before booking.");
499.                     Optional<ButtonType> answer = alert.showAndWait();
500.
501.
502.                     if(answer.get() == ButtonType.OK){
503.                         cc.setIsSelectedSeat(true);
504.                         parent = FXMLLoader.load(getClass().getResource("/cinema
    /ui/auth/Login.fxml"));
505.                         Scene parentScene = new Scene(parent);
506.                         Stage window = (Stage)((Node)event.getSource()).getScene
    ().getWindow();
```

```
507.                             window.setScene(parentScene);
508.                             window.show();
509.                         }
510.                         else {
511.                             return;
512.                         }
513.
514.                     }
515.
516.                 }
517.
518.
519.                 @FXML
520.                 private void handleButtonAction(javafx.scene.input.MouseEvent event)
     throws FileNotFoundException { //MouseEvent
521.                     ImageView iv = (ImageView) event.getSource(); // get image sourc
    e
522.                     InputStream fileInputStream;
523.                     isFound = false;
524.                     // Loop to find id is list
525.                     for (String id : ivList) {
526.                         if (id.equals(iv.getId())) {
527.                             isFound = true;
528.                         }
529.                     }
530.                     if (isFound) {
531.                         if (Integer.parseInt(maxIV.substring(1)) == Integer.parseInt
    (iv.getId().substring(1)) || Integer.parseInt(iv.getId().substring(1)) == Integer.p
    arseInt(minIV.substring(1))) {
532.                             System.out.println(maxIV.substring(1));
533.                             if (Integer.parseInt(maxIV.substring(1)) == Integer.pars
    eInt(iv.getId().substring(1))) {
534.                                 maxIV = maxIV.substring(0, 1) + Integer.toString(Int
    eger.parseInt(maxIV.substring(1)) - 1);
535.
536.                             System.out.println(maxIV);
537.
538.
539.                             } else {
540.                                 minIV = minIV.substring(0, 1) + Integer.toString(Int
    eger.parseInt(minIV.substring(1)) + 1);
541.                                 System.out.println(minIV);
542.
543.
544.                             }
545.                             if (Integer.parseInt(maxIV.substring(1)) < Integer.parse
    Int(minIV.substring(1))) {
546.                                 count = 0;
547.                             }
548.                             //                System.out.println("Found"); // เจอข้อมูล
549.                             ivList.remove(new String(iv.getId())); // ลบ id นั้นออก
550.                             if (new String(minIV.substring(0, 1)).equals("A")) {
551.                                 fileInputStream = getClass().getResourceAsStream("/c
    inema/ui/images/pair.png");
552.                                 Image image = new Image(fileInputStream); //, 100, 2
    00, false, true
553.                                 iv.setImage(image);
554.
555.
556.                             } else if (new String(minIV.substring(0, 1)).equals("C")
     || new String(minIV.substring(0, 1)).equals("B")) {
557.                                 fileInputStream = getClass().getResourceAsStream("/c
    inema/ui/images/blue.png");
558.                                 Image image = new Image(fileInputStream); //, 100, 2
    00, false, true
```

```
559.                                 iv.setImage(image);
560.                             } else {
561.                                 fileInputStream = getClass().getResourceAsStream("/c
    inema/ui/images/red.png");
562.                                 Image image = new Image(fileInputStream); //, 100, 2
    00, false, true
563.                                 iv.setImage(image);
564.                             };
565.
566.
567.                         } else {
568.                             AlertMaker.showErrorMessage("แจ้งเตือน", "ไม่สามารถลบที่นั่งที่ไม่ได้อยู่ชิดริม
    ได้");
569.                             System.out.print("can't");
570.                         }
571.                     } else {
572.                         if (count > 0) {
573.
574.
575.
576.                         if (new String(minIV.substring(0, 1)).equals(iv.getId().subs
    tring(0, 1))) { //check same line
577.                             if (Integer.parseInt(maxIV.substring(1)) + 1 == Inte
    ger.parseInt(iv.getId().substring(1)) || Integer.parseInt(iv.getId().substring(1))
    == Integer.parseInt(minIV.substring(1)) - 1) { //check continuous
578.                                 if (count == 1) {
579.                                     if (Integer.parseInt(iv.getId().substring(1)
    ) > Integer.parseInt(iv.getId().substring(1))) {
580.                                         maxIV = minIV;
581.                                         minIV = iv.getId();
582.                                     } else {
583.                                         maxIV = iv.getId();
584.                                     }
585.
586.
587.                                 } else {
588.                                     if (Integer.parseInt(iv.getId().substring(1)
    ) == Integer.parseInt(maxIV.substring(1)) + 1) {
589.                                         maxIV = iv.getId();
590.                                     } else {
591.                                         minIV = iv.getId();
592.                                     }
593.
594.
595.                                 }
596.                                 // check that minIV > maxIV, if yes, swap!!
597.                                 if (Integer.parseInt(minIV.substring(1)) > Integ
    er.parseInt(maxIV.substring(1))) {
598.                                     String temp = maxIV;
599.                                     maxIV = minIV;
600.                                     minIV = temp;
601.                                 }
602.
603.
604.                                 // System.out.println("Not found"); // ไม่เจอข้อมูล
605.                                 ivList.add(iv.getId()); // เพิ่ม id เข้าไป
606.                                 if (new String(minIV.substring(0, 1)).equals("A"
    )) {
607.                                     fileInputStream = getClass().getResourceAsSt
    ream("/cinema/ui/images/pair2.png");
608.                                     Image image = new Image(fileInputStream); //
    , 100, 200, false, true
609.                                     iv.setImage(image);
610.
611.
```

```
612.
613.                                 } else if (new String(minIV.substring(0, 1)).equals(
    "C") || new String(minIV.substring(0, 1)).equals("B")) {
614.                                     fileInputStream = getClass().getResourceAsSt
    ream("/cinema/ui/images/blue2.png");
615.                                 Image image = new Image(fileInputStream); //, 100, 2
    00, false, true
616.                                 iv.setImage(image);
617.                                     } else {
618.                                     fileInputStream = getClass().getResourceAsSt
    ream("/cinema/ui/images/red2.png");
619.                                     Image image = new Image(fileInputStream); //
    , 100, 200, false, true
620.                                     iv.setImage(image);
621.                                 };
622.
623.
624.                                 } else {
625.                                     AlertMaker.showErrorMessage("แจ้งเตือน", "กรุณาเลือกที่นั่งที่
    ติดกัน"); //+minIV+maxIV);
626.                                     System.out.print("Noooo continuous");
627.                                 }
628.                             } else {
629.                                 AlertMaker.showErrorMessage("แจ้งเตือน", "กรุณาเลือกที่นั่งแถว
    เดียวกัน");
630.                                 System.out.print("Nooooooooo same line");
631.                             }
632.                         } else {
633.                             System.out.print(iv.getId().substring(0, 1));
634.                             minIV = iv.getId();
635.                             maxIV = iv.getId();
636.                             System.out.println(new String(minIV.substring(0, 1)).equ
    als("C"));
637.                             //                System.out.println("Not found"); // ไม่
    เจอข้อมูล
638.                             ivList.add(iv.getId()); // เพิ่ม id เข้าไป
639.                             if (new String(minIV.substring(0, 1)).equals("A")) {
640.                                 fileInputStream = getClass().getResourceAsStream("/c
    inema/ui/images/pair2.png");
641.                                 Image image = new Image(fileInputStream); //, 100, 2
    00, false, true
642.                                 iv.setImage(image);
643.
644.
645.                             } else if (new String(minIV.substring(0, 1)).equals("C")
     || new String(minIV.substring(0, 1)).equals("B")) {
646.                                 fileInputStream = getClass().getResourceAsStream("/c
    inema/ui/images/blue2.png");
647.
648.                             Image image = new Image(fileInputStream); //, 100, 200,
    false, true
649.                                 iv.setImage(image);
650.                             } else {
651.                                 fileInputStream = getClass().getResourceAsStream("/c
    inema/ui/images/red2.png");
652.                                 Image image = new Image(fileInputStream); //, 100, 2
    00, false, true
653.                                 iv.setImage(image);
654.                             };
655.
656.
657.                         }
658.                         count += 1;
659.                     }
660.
```

```
661.                     System.out.println(ivList);
662.                     if(ivList.size() > 0){
663.                         double sum = 0;
664.                         String type = "";
665.                         double seatPrice = 0;
666.                         Seat s = cc.getShowtimeSeat(cc.getSelectShowtime(),ivList.ge
    t(0));
667.                         type = s.getSeatType();
668.                         seatPrice = s.getSeatPrice();
669.
670.                         sum = ivList.size() * seatPrice;
671.             //            for (String seatName : ivList) {
672.             //                s = cc.getShowtimeSeat(cc.getSelectShowtime(),seatName
    );
673.             //                sum += s.getSeatPrice();
674.             //            }
675.                         labelPreSummary.setText("ที่นั่ง
    ประเภท : " + type + " Seat  จำนวน : " + ivList.size() + " ที่ x " + seatPrice + " = "  +
    sum + " บาท");
676.                         // Set seat to controller;
677.                         if(uc.getIsLogin()){
678.                             btnNext.setDisable(false);
679.                         }else{
680.                             btnNext.setDisable(false);
681.                         }
682.
683.                     }else{
684.                         labelPreSummary.setText(null);
685.                         btnNext.setDisable(true);
686.                     }
687.
688.
689.                 }
690.
691.
692.                 @FXML
693.                 private void back(ActionEvent event) throws IOException {
694.                     Parent parent;
695.                     parent = FXMLLoader.load(getClass().getResource("/cinema/ui/show
    time/showtime.fxml"));
696.                     Scene parentScene = new Scene(parent);
697.                     Stage window = (Stage)((Node)event.getSource()).getScene().getWi
    ndow();
698.                     window.setScene(parentScene);
699.                     window.show();
700.                 }
701.             }
```

หน้าสรุปรายการการชำระเงินของสมาชิก

Class summaryController

```
1.  package cinema.ui.summary;
2.
3.  import cinema.Booking;
4.  import cinema.CinemaController;
5.  import cinema.Movie;
6.  import cinema.NormalSeat;
7.  import cinema.Promotion;
8.  import cinema.PromotionController;
9.  import cinema.Seat;
10. import cinema.Showtime;
11. import cinema.Theatre;
12. import cinema.User;
13. import cinema.UserController;
14. import cinema.ui.AlertMaker;
15. import java.io.IOException;
16. import java.net.URL;
17. import java.time.LocalDate;
18. import java.time.ZoneId;
19. import java.time.format.DateTimeFormatter;
20. import static java.time.temporal.TemporalQueries.localDate;
21. import java.util.ArrayList;
22. import java.util.Date;
23. import java.util.List;
24. import java.util.ResourceBundle;
25. import javafx.beans.value.ChangeListener;
26. import javafx.beans.value.ObservableValue;
27. import javafx.collections.FXCollections;
28. import javafx.collections.ObservableList;
29. import javafx.event.ActionEvent;
30. import javafx.fxml.FXML;
31. import javafx.fxml.FXMLLoader;
32. import javafx.fxml.Initializable;
33. import javafx.scene.Node;
34. import javafx.scene.Parent;
35. import javafx.scene.Scene;
36. import javafx.scene.control.Button;
37. import javafx.scene.control.ComboBox;
38. import javafx.scene.layout.StackPane;
39. import javafx.scene.text.Text;
40. import javafx.stage.Stage;
41.
42. /**
43.  * FXML Controller class
44.  *
45.  * @author BEAMCONAN
46.  */
47. public class summaryController implements Initializable {
48.
49.     @FXML
50.     private Text nameEng;
51.     @FXML
52.     private Text theatre;
53.     @FXML
54.     private Text date;
55.     @FXML
56.     private Text typeSeat;
57.     @FXML
58.     private Text amountSeat;
59.     @FXML
```

```
60.     private Text costPerSeat;
61.     @FXML
62.     private Text cost;
63.     @FXML
64.     private ComboBox<Promotion> selectPromotion;
65.     @FXML
66.     private Text sumcost;
67.     @FXML
68.     private Text moneyUser;
69.     @FXML
70.     private Text discount;
71.     @FXML
72.     private Text details;
73.
74.     @FXML
75.     private Text username;
76.     @FXML
77.     private Text showtime;
78.     @FXML
79.     private StackPane rootPane;
80.
81.     User user;
82.     Booking booking;
83.     Movie movie;
84.
85.     CinemaController cc;
86.     UserController uc;
87.     PromotionController pc;
88.     User loginUser;
89.     Promotion userSelectPromotion = null;
90.
91.     @FXML
92.     private Button btnCancelPromotion;
93.
94.     public summaryController() {
95.         this.uc = uc.getInstance();
96.         this.pc = pc.getInstance();
97.         this.cc = cc.getInstance();
98.     }
99.
100.          Double totalPrice;
101.          Showtime st;
102.          List<Seat> seatList;
103.
104.          @Override
105.          public void initialize(URL url, ResourceBundle rb) {
106.              if(uc.getIsLogin()){
107.                  loginUser = uc.getUser(uc.getLoginUser().getId()); // update dat
    a
108.                  //System.out.println(loginUser.getUsername());
109.              }else{
110.                  System.err.println("Please login");
111.              }
112.
113.              st = cc.getShowtime(cc.getSelectShowtime()); // get showtime
114.              seatList = cc.getSeatList(); // get list of user select seat from la
    yout
115.              totalPrice = seatList.size() * seatList.get(0).getSeatPrice();
116.      //*****************************Details movie*********************
117.              nameEng.setText(st.getMovieEng());
118.              theatre.setText(st.getTheatreNo() + " (" + st.getSystem() + ") " + s
    t.getSoundtrack().toUpperCase() + "/" + st.getSubtitle().toUpperCase());
119.      //***************************** get today *************************
120.              date.setText(st.getDate());
121.              showtime.setText(st.getShowtime());
122.              typeSeat.setText(seatList.get(0).getSeatType());
```

```
123.                amountSeat.setText(String.valueOf(seatList.size()));
124.                costPerSeat.setText(String.valueOf(seatList.get(0).getSeatPrice()));

125.                cost.setText(Double.toString(totalPrice));
126.
127.                loadPromotionSelect();
128.
129.                discount.setText(Double.toString(0));
130.                details.setText(null);
131.                sumcost.setText(Double.toString(totalPrice));
132.                btnCancelPromotion.setVisible(false);
133.                userSelectPromotion = null;
134.        //
135.        //*****************************Details user********************
136.                username.setText(loginUser.getFirstname());
137.                moneyUser.setText(Double.toString(uc.getUser(loginUser.getId()).getM
    oney()));
138.        ////****************************** test Promotion ********************
139.
140.            }
141.
142.            public void loadPromotionSelect(){
143.                ObservableList<Promotion> promotionList = FXCollections.observableAr
    rayList();
144.                promotionList.clear();
145.                List<Promotion> allPromotion = pc.getActivePromotionList();
146.                List<Promotion> usedPromotion = uc.getUser(loginUser.getId()).getPro
    motionList(); // update get user promotion from id
147.
148.                // Get never used promotion
149.                if(usedPromotion.size() > 0){
150.                    allPromotion = pc.getUnusePromotion(loginUser.getId());
151.                    if(allPromotion.size() > 0){
152.                        selectPromotion.setDisable(false);
153.                        System.err.println(allPromotion.size());
154.                        for (Promotion promotion : allPromotion) {
155.                            promotionList.add(promotion);
156.                        }
157.                        selectPromotion.setItems(promotionList);
158.                    }else{
159.                        selectPromotion.setDisable(true);
160.                    }
161.                }else{
162.                    if(allPromotion.size() > 0){
163.                        selectPromotion.setDisable(false);
164.                        for (Promotion aP : allPromotion) {
165.                             promotionList.add(aP);
166.                              System.err.println("USER NO PROMOTION");
167.                        }
168.                        selectPromotion.setItems(promotionList);
169.                    }else{
170.                        selectPromotion.setDisable(true);
171.                    }
172.
173.                }
174.
175.
176.                selectPromotion.setStyle("-fx-text-fill: #ffffff;");
177.                //selectPromotion.setValue("Don't use promotion");
178.
179.            }
180.
181.            public void topupUserMoney(Double money){
182.                uc.topupUserMoney(loginUser.getId(), money);
183.                moneyUser.setText(Double.toString(uc.getUserMoney(loginUser.getId())
    ));
```

```
184.            }
185.            @FXML
186.            private void plusMoney100(ActionEvent event) {
187.                topupUserMoney(100.0);
188.            }
189.            @FXML
190.            private void plusMoney150(ActionEvent event) {
191.                topupUserMoney(150.0);
192.            }
193.            @FXML
194.            private void plusMoney200(ActionEvent event) {
195.                topupUserMoney(200.0);
196.            }
197.            @FXML
198.            private void plusMoney300(ActionEvent event) {
199.                topupUserMoney(300.0);
200.            }
201.            @FXML
202.            private void plusMoney500(ActionEvent event) {
203.                topupUserMoney(500.0);
204.            }
205.            @FXML
206.            private void plusMoney1000(ActionEvent event) {
207.                topupUserMoney(1000.0);
208.            }
209.
210.            @FXML
211.            private void payment(ActionEvent event) throws IOException {
212.                //booking.payment();
213.                System.err.print("Start Booking...");
214.                double total = Double.valueOf(sumcost.getText());
215.                if(uc.getUser(loginUser.getId()).getMoney() >= total){
216.                    Booking b = new Booking(st, seatList, uc.getUser(loginUser.getId
    ()), userSelectPromotion, total);
217.                    System.out.println(b);
218.                    cc.addBooking(b);
219.                    AlertMaker.showSimpleAlert("ซื้อที่นั่งสำเร็จ", "เหลือ
    เงิน : " + uc.getUser(loginUser.getId()).getMoney());
220.                    Parent parent;
221.                    parent = FXMLLoader.load(getClass().getResource("/cinema/ui/show
    movie/showmovie.fxml"));
222.                    Scene parentScene = new Scene(parent);
223.                    Stage window = (Stage)((Node)event.getSource()).getScene().getWi
    ndow();
224.                    window.setScene(parentScene);
225.                    window.show();
226.                }else{
227.                    AlertMaker.showErrorMessage("จำนวนเงินไม่เพียงพอ","กรุณาเติมเงินเข้าสู่ระบบ");
228.                }
229.            }
230.
231.            @FXML
232.            private void back(ActionEvent event) throws IOException {
233.                Parent parent;
234.                parent = FXMLLoader.load(getClass().getResource("/cinema/ui/layoutMe
    dium/layoutMedium.fxml"));
235.                Scene parentScene = new Scene(parent);
236.                Stage window = (Stage)((Node)event.getSource()).getScene().getWindow
    ();
237.                window.setScene(parentScene);
238.                window.show();
239.            }
240.
241.            @FXML
242.            private void handlePromotion(ActionEvent event) {
```

```
243.                // loadPromotionSelect();
244.                 System.out.println("Hello Promotion");
245.                     if(selectPromotion.getValue() != null){
246.                     Promotion p = selectPromotion.getValue();
247.                     discount.setText(Double.toString(p.getDiscount()));
248.                     details.setText(p.getDescription());
249.                     Double lastTotal = totalPrice - p.getDiscount();
250.                     if(lastTotal >= 0){
251.                         sumcost.setText(Double.toString(totalPrice - p.getDiscount()
    ));
252.                     }else{
253.                         sumcost.setText("0.0");
254.                     }
255.                     btnCancelPromotion.setVisible(true);
256.                     userSelectPromotion = p;
257.                 }
258.             }
259.
260.             @FXML
261.             private void handleCancelPromotion(ActionEvent event) {
262.                 discount.setText(Double.toString(0));
263.                 details.setText(null);
264.                 sumcost.setText(Double.toString(totalPrice));
265.                 selectPromotion.setValue(null);
266.                 btnCancelPromotion.setVisible(false);
267.                 userSelectPromotion = null;
268.             }
269.
270.
271.         }
```

หน้าสรุปรายการการชำระเงินของพนักงาน

Class summaryStaffController

```
1.  package cinema.ui.summaryStaff;
2.
3.  import cinema.Booking;
4.  import cinema.CinemaController;
5.  import cinema.Movie;
6.  import cinema.Promotion;
7.  import cinema.PromotionController;
8.  import cinema.Seat;
9.  import cinema.Showtime;
10. import cinema.User;
11. import cinema.UserController;
12. import cinema.ui.AlertMaker;
13. import java.io.IOException;
14. import java.net.URL;
15. import java.util.List;
16. import java.util.ResourceBundle;
17. import javafx.event.ActionEvent;
18. import javafx.fxml.FXML;
19. import javafx.fxml.FXMLLoader;
20. import javafx.fxml.Initializable;
21. import javafx.scene.Node;
22. import javafx.scene.Parent;
23. import javafx.scene.Scene;
24. import javafx.scene.layout.StackPane;
25. import javafx.scene.text.Text;
26. import javafx.stage.Stage;
27.
28. /**
29.  * FXML Controller class
30.  *
31.  * @author BEAMCONAN
32.  */
33. public class summaryStaffController implements Initializable {
34.
35.     @FXML
36.     private Text nameEng;
37.     @FXML
38.     private Text theatre;
39.     @FXML
40.     private Text date;
41.     @FXML
42.     private Text typeSeat;
43.     @FXML
44.     private Text amountSeat;
45.     @FXML
46.     private Text costPerSeat;
47.     @FXML
48.     private Text cost;
49.     @FXML
50.     private Text sumcost;
51.     @FXML
52.     private Text showtime;
53.     @FXML
54.     private StackPane rootPane;
55.
56.     User user;
57.     Booking booking;
58.     Movie movie;
59.
```

```
60.     CinemaController cc;
61.     UserController uc;
62.     User loginUser;
63.     Promotion userSelectPromotion = null;
64.
65.     public summaryStaffController() {
66.         this.uc = uc.getInstance();
67.         this.cc = cc.getInstance();
68.     }
69.
70.     Double totalPrice;
71.     Showtime st;
72.     List<Seat> seatList;
73.
74.     @Override
75.     public void initialize(URL url, ResourceBundle rb) {
76.          if(uc.getIsLogin()){
77.             loginUser = uc.getUser(uc.getLoginUser().getId()); // update data
78.             //System.out.println(loginUser.getUsername());
79.         }else{
80.             System.err.println("Please login");
81.         }
82.
83.         st = cc.getShowtime(cc.getSelectShowtime()); // get showtime
84.         seatList = cc.getSeatList(); // get list of user select seat from layout
85.         totalPrice = seatList.size() * seatList.get(0).getSeatPrice();
86.
87. //*****************************Details movie*********************
88.         nameEng.setText(st.getMovieEng());
89.         theatre.setText(st.getTheatreNo() + " (" + st.getSystem() + ") " + st.getSo
    undtrack().toUpperCase() + "/" + st.getSubtitle().toUpperCase());
90. //***************************** get today **************************
91.         date.setText(st.getDate());
92.         showtime.setText(st.getShowtime());
93.         typeSeat.setText(seatList.get(0).getSeatType());
94.         amountSeat.setText(String.valueOf(seatList.size()));
95.         costPerSeat.setText(String.valueOf(seatList.get(0).getSeatPrice()));
96.         cost.setText(Double.toString(totalPrice));
97.         sumcost.setText(Double.toString(totalPrice));
98.     }
99.
100.            @FXML
101.            private void payment(ActionEvent event) throws IOException {
102.                // ทำการจอง
103.                System.err.println("Staff Booking");
104.                double total = Double.valueOf(sumcost.getText());
105.                    Booking b = new Booking(st, seatList, uc.getUser(loginUser.getId
    ()), userSelectPromotion, total);
106.                    System.out.println(b);
107.
108.                    cc.addStaffBooking(b);
109.                    AlertMaker.showSimpleAlert("OK", "Book complete");
110.                    Parent parent;
111.                    parent = FXMLLoader.load(getClass().getResource("/cinema/ui/myBo
    oking/MyBooking.fxml"));
112.                    Scene parentScene = new Scene(parent);
113.                    Stage window = (Stage)((Node)event.getSource()).getScene().getWi
    ndow();
114.                    window.setScene(parentScene);
115.                    window.show();
116.            }
117.
118.            @FXML
119.            private void back(ActionEvent event) throws IOException {
120.                Parent parent;
```

```
121.                parent = FXMLLoader.load(getClass().getResource("/cinema/ui/layoutMe
    dium/layoutMedium.fxml"));
122.                Scene parentScene = new Scene(parent);
123.                Stage window = (Stage)((Node)event.getSource()).getScene().getWindow
    ();
124.                window.setScene(parentScene);
125.                window.show();
126.            }
127.
128.
129.        }
```