# บทที่ 4

**การพัฒนาระบบงาน**

ในบทนี้จะกล่าวถึงการสร้างระบบงานของเว็บแอปพลิเคชันระบบรวมอินดี้อาร์ทอุบล โดยอธิบายถึงลักษณะการทำงานของระบบต่าง ๆ ในเว็บแอปพลิเคชัน พร้อมยกตัวอย่างโปรแกรม โดยแบ่งเป็นส่วน ดังนี้

4.1 การพัฒนาในส่วนการอัพโหลด

4.2 การพัฒนาในส่วนการตั้งกระทู้

4.3 การพัฒนาในส่วนการเรียกดูข้อมูล

4.4 การพัฒนาในส่วนการแก้ไขข้อมูล

4.5 การพัฒนาในส่วนการลบข้อมูล

4.6 การพัฒนาในส่วนการแสดงความคิดเห็น

4.7 การพัฒนาในส่วนการติดตามสมาชิก

4.8 การพัฒนาในส่วนแฟ้มสะสมผลงาน

## 4.1 การพัฒนาในส่วนการอัพโหลด

เมื่อสมาชิกต้องการอัพโหลดผลงานสามมิติ ระบบส่งหน้าไปยังฟอร์ม เพื่อทำการกรอกข้อมูลผลงาน และทำการอัพโหลดผลงาน ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 5 ไฟล์ ได้แก่ IndieApplication.scala, ApplicationController.scala, ArtWork.scala, upload.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การสร้างไฟล์ html โดยชื่อว่า upload.scala.html สำหรับการแสดงผล

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | @import b3.inline.fieldConstructor  @blend = @{ Seq("Category"->"Category","Character"->"Character","Assets"->"Assets","Scene"->"Scene","Others"->"Others") }  <div class="col-sm-8 “>  @helper.form(action = routes.IndieApplication.upload,'enctype -> "multipart/form-data") {  @helper.CSRF.formField  @b3.text(uploadForm("title"), '\_label -> "Topic")  @b3.textarea( uploadForm("detail"), '\_label -> "details" ,'rows -> 3 )  @b3.select( uploadForm("tags"), options = blend, '\_label -> "Select" )  @b3.file( uploadForm("picture"), '\_label -> "Picturefile" )  @b3.file( uploadForm("fileblend"), '\_label -> "Blendfile" )  @b3.file( uploadForm("filehtml"), '\_label -> "Htmlfile" )  <div class=" text-right">  <button type="submit" class="btn btn-primary">Submit</button>  &nbsp; &nbsp;  <button type="submit" class="btn btn-danger">Cancel</button>  </div>  } |

ภาพที่ 4.1 หน้าหลักการอัพโหลดผลงานในส่วน View

จากภาพที่ 4.1 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 6 คือช่องสำหรับกรอกข้อมูลหัวข้อ

บรรทัดที่ 7 คือช่องสำหรับกรอกรายละเอียดเกี่ยวกับผลงาน

บรรทัดที่ 8 คือช่องสำหรับเลือกหมวดหมู่ของผลงาน

บรรทัดที่ 9 คือช่องสำหรับอัพโหลดไฟล์รูปภาพ

บรรทัดที่ 10 คือช่องสำหรับอัพโหลดไฟล์.blend

บรรทัดที่ 11 คือช่องสำหรับอัพโหลดไฟล์ html

บรรทัดที่ 13 คือปุ่มสำหรับส่งข้อมูลการอัพโหลดซึ่งจะเรียกใช้ฟังก์ชัน upload ในIndieApplication.scala

บรรทัดที่ 14 คือปุ่มสำหรับยกเลิกการอัพโหลด และกลับสู่หน้าหลักของระบบ

1. สร้างไฟล์ Controller ที่มีชื่อว่า IndieApplication.scala ขึ้นมาเพื่อใช้ในการควบคุมการทำงาน และกำหนดการแสดงผลของ View

**ภาพที่ 4.2** หน้าหลักการอัพโหลดผลงานในส่วน Controller

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | defdef upload = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  //Logger.warn(s"IndieApplication.upload(user=${user.userID}, uuid=${uuid})")  val defaults = Seq("")  val uuid = UUID.randomUUID  request.body.asMultipartFormData.map { a =>  for (  title <- a.dataParts.get("title");  detail <- a.dataParts.get("detail");  tags <- a.dataParts.get("tags");  pictureFile <- a.file("picture");  blendFile <- a.file("fileblend");  htmlFile <- a.file("filehtml")  ) yield {  //Logger.warn(s"pic = ${pic}, blend = ${blend}, html = ${html}")  val pictureExtension = reflect.io.File(pictureFile.filename).extension  val picture = s"$uuid.$pictureExtension"  val fileblend = s"$uuid.blend"  val filehtml = s"$uuid.html"  val dbartwork = ArtWork (  //id = UUID.randomUUID.toString,  id = uuid.toString,  userID = user.userID.toString,  title = title(0),  detail = detail(0),  tags = tags(0),  picture = picture,  fileblend = fileblend,  filehtml = filehtml  )  uploadart.add(dbartwork) // save to db  // move files  pictureFile.ref.moveTo(new File(s"public/members/${user.userID}/$picture"))  blendFile.ref.moveTo(new File(s"public/members/${user.userID}/$fileblend"))  htmlFile.ref.moveTo(new File(s"public/members/${user.userID}/$filehtml"))  }  Future.successful(Redirect("/model"))  }.getOrElse {  Future.successful(Redirect("/up"))  }  case None => Future.successful(Redirect("/"))  }  } |

**ภาพที่ 4.2** **(ต่อ)** หน้าหลักการอัพโหลดผลงานในส่วน Controller

จากภาพที่ 4.2 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-2 ฟังก์ชัน upload() คือการอัพโหลดผลงานของสมาชิกในระบบ โดยจะมีการตรวจสอบสิทธิ์

ผู้ใช้งาน ซึ่งจะรียกใช้ไฟล์ User.scala เมื่อสิทธิ์ผู้ใช้ถูกต้องจึงจะสามารถทำการอัพโหลดได้

บรรทัดที่ 3-51 คือ การแปลงข้อมูลจาก Object ให้อยู่ในรูปแบบ String เพื่อบันทึกข้อมูลลงฐานข้อมูล

บรรทัดที่ 53-65 dbartwork คือการส่งข้อมูลไปยังไฟล์ ArtWork.scala โดยเรียกใช้ฟังก์ชัน saveDate

เพื่อเพิ่มข้อมูลการอัพโหลดลงในฐานข้อมูล โดยส่งข้อมูลไปยัง Object uploadart มีการเรียกใช้ ฟังก์ชัน add และทำการบันทึกข้อมูลลงตาราง artwork ในระบบฐานข้อมูล

บรรทัดที่ 66-69 เมื่อการตรวจสอบข้อมูลถูกต้อง ข้อมูลจะถูงส่งไฟยังฟังก์ชัน uploadfile() ในไฟล์

ApplicationController.scala เพื่อทำการอัพโหลดไฟล์

บรรทัดที่ 70 คือ case เมื่อไม่พบสิทธิ์ผู้ใช้งาน จะส่งค่ากลับไปหน้าหลัก

บรรทัดที่ 73-76 ฟังก์ชัน getdata คือการแปลง Option ให้เป็น String โดยถูกเรียกใช้บรรทัดที่ 3-51

1. สร้างไฟล์ Models ที่มีชื่อว่า ArtWork.scala ขึ้นมาเพื่อใช้ในส่วนของการจัดการฐานข้อมูล และติดต่อกับส่วนการควบคุมการทำงาน ของ Controller

**ภาพที่ 4.3** หน้าหลักการอัพโหลดผลงานในส่วน Models

**ภาพที่ 4.3** หน้าหลักการอัพโหลดผลงานในส่วน Models

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | case class ArtWork(  id: String,  userID: String,  title: String,  detail: String,  tags: String,  picture: String,  fileblend: String,  filehtml: String  )  class artworks(tag: Tag) extends Table[ArtWork](tag, "artwork") {  def id = column[String]("id", O.PrimaryKey)  def userID = column[String]("userID")  def title = column[String]("title")  def detail = column[String]("detail")  def tags = column[String]("tags")  def picture = column[String]("picture")  def fileblend = column[String]("fileblend")  def filehtml = column[String]("filehtml")  // def date = column[Option[java.sql.Date]]("date")  def \* = ( id, userID, title, detail, tags, picture,fileblend , filehtml ) <> (ArtWork.tupled, ArtWork.unapply)  }  object uploadart {  val dbConfig = DatabaseConfigProvider.get[JdbcProfile](Play.current)  val dbupload = TableQuery[artworks]    def add(de: ArtWork): Future[String] = {  dbConfig.db.run(dbupload += de).map(res => "successfully").recover {  case ex: Exception => ex.getCause.getMessage  }  } |

**ภาพที่ 4.3 (ต่อ)** หน้าหลักการอัพโหลดผลงานในส่วน Models

จากภาพที่ 4.3 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-10 คือคลาส Artwork ที่เก็บค่าไว้เพื่อเชื่อมต่อกับฐานข้อมูล

บรรทัดที่ 12-23 คือการ Mapping ข้อมูลลงในตาราง artwork ในฐานข้อมูลของระบบ

บรรทัดที่ 26 คือการดึงข้อมูลจากตารางมาใช้

บรรทัดที่ 28-32 คือ การเรียกใช้ฟังก์ชัน add เพื่อทำการเพิ่มข้อมูลลงฐานข้อมูล

1. สร้างฟังก์ชัน ชื่อว่า uploadfile() ในไฟล์ ApplicationController.scala ขึ้นมาเพื่อควบคุมการทำงานการอัพโหลดไฟล์

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | def uploadfile = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  val c = for{  a <- ListUser.get(user.userID.toString)  }yield a  c.map { case (role) =>  Ok(views.html.uploadModel(user,uploadForm.form))  }  case None => Future.successful(Redirect("/"))  }  } |

**ภาพที่ 4.4** หน้าหลักการอัพโหลดผลงานในส่วน Controller

จากภาพที่ 4.3 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-3 ฟังก์ชัน uploadfile() คือการอัพโหลดผลงานของสมาชิกในระบบ โดยจะมีการตรวจสอบ

สิทธิ์ผู้ใช้งาน

บรรทัดที่ 4-9 ทำการดึงข้อมูลของผู้ใช้ และส่งข้อมูลไปยังหน้า View

บรรทัดที่ 10 คือ case เมื่อไม่พบสิทธิ์ผู้ใช้งาน จะส่งค่ากลับไปหน้าหลัก

1. การควบคุมการทำงานฟังกืชันอัพโหลดผลงาน ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | GET /up controllers.ApplicationController.uploadfile |

**ภาพที่ 4.5** การจัดการอัพโหลดผลงานในส่วน Routes

จากภาพที่ 4.3 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ GET ผ่าน URL: /up ซึ่งมีการเรียกใช้ฟังก์ชัน uploadfile() โดย

Controllers มีชื่อไฟล์ว่า ApplicationController

## 4.2 การพัฒนาในส่วนการตั้งกระทู้

เมื่อสมาชิกต้องการตั้งกระทุ้สนทนา ระบบจะส่งหน้าไปยังฟอร์ม เพื่อทำการกรอกข้อมูลกระทู้สนทนา และทำการบันทึกการตั้งกระทู้ ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ IndieApplication.scala, Foruminfo.scala, createPosts.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การสร้างไฟล์ html โดยชื่อว่า createPosts.scala.html สำหรับการแสดงผล

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @import b3.inline.fieldConstructor  <div class="col-sm-8 “>  @helper.form(action = routes.IndieApplication.forum,'enctype -> "multipart/form-data") {  @helper.CSRF.formField  <h5 >หัวข้อกระทู้</h5>  @b3.text( forumForm("title"), '\_label -> "Topic" )  @b3.textarea( forumForm("detail"), '\_label -> "details" ,'rows -> 3 )  @b3.file( forumForm("picture"), '\_label -> "File" )  <div class=" text-right">  <button type="submit" class="btn btn-primary">Submit</button> &nbsp; &nbsp;  <button type="submit" class="btn btn-danger">Cancel</button>  </div>  } |

ภาพที่ 4.6 หน้าหลักของการตั้งกระทู้สนทนาในส่วน View

จากภาพที่ 4.6 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 6 คือช่องสำหรับกรอกข้อมูลหัวข้อกระทู้

บรรทัดที่ 7 คือช่องสำหรับกรอกรายละเอียดกระทู้

บรรทัดที่ 8 คือช่องสำหรับอัพโหลดไฟล์รูปภาพ

บรรทัดที่ 10 คือปุ่มสำหรับส่งข้อมูลการตั้งกระทู้ซึ่งจะเรียกใช้ฟังก์ชัน forum ในไฟล์

IndieApplication.scala

บรรทัดที่ 11 คือปุ่มสำหรับยกเลิกการตั้งกระทู้ และกลับสู่หน้าหลักของระบบ

1. สร้างฟังก์ชัน forum ในไฟล์ที่มีชื่อว่า IndieApplication.scala ขึ้นมาเพื่อใช้ในการจัดการตั้งกระทู้ และกำหนดการแสดงผลไปยัง View

**ภาพที่ 4.7** หน้าหลักการตั้งกระทู้สนทนาในส่วน Controller

**ภาพที่ 4.7** หน้าหลักการตั้งกระทู้สนทนาในส่วน Controller

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | def forum = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  request.body.asMultipartFormData.map {a =>  val datatitle = a.dataParts.get("title").map { a =>  for{  b <- a.mkString("")  }yield b  }  val datadetail = a.dataParts.get("detail").map { a =>  for{  b <- a.mkString("")  }yield b  }  val dataimg = a.file("picture").map { a=>  val filename = a.filename  val extension = FilenameUtils.getExtension(filename)  val newFileName = s"${UUID.randomUUID}.$extension"  a.ref.moveTo(new File(s"public/images/$newFileName"))  for{  b <- newFileName  }yield b  }  val title = getdata(datatitle)  val detail = getdata(datadetail)  val picture = getdata(dataimg)  //add table database  val dbforum = Foruminfo (  id = UUID.randomUUID.toString,  userID = user.userID.toString,  title = title ,  detail = detail ,  picture = picture  )  val saveDate = for{  a <- addforum.add(dbforum)  }yield a  Future.successful(Redirect("/forums"))  }.getOrElse {  Future.successful(Redirect("/forums"))  }  case None => Future.successful(Redirect("/"))  }  }  **ภาพที่ 4.7 (ต่อ)** หน้าหลักการตั้งกระทู้สนทนาในส่วน Controller |

จากภาพที่ 4.7 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-2 ฟังก์ชัน forum() คือการตั้งกระทู้สนทนาของสมาชิกในระบบ โดยจะมีการตรวจสอบสิทธิ์

ผู้ใช้งาน ซึ่งจะรียกใช้ไฟล์ User.scala เมื่อสิทธิ์ผู้ใช้ถูกต้องจึงจะสามารถทำการตั้งกระทู้ได้

บรรทัดที่ 3-25 การแปลงข้อมูลจาก Object ให้อยู่ในรูปแบบ String เพื่อบันทึกข้อมูลลงฐานข้อมูล

บรรทัดที่ 28-37 dbforum คือการส่งข้อมูลไปยังไฟล์ Foruminfo.scala โดยเรียกใช้ฟังก์ชัน saveDate

เพื่อเพิ่มข้อมูลการตั้งกระทู้ในฐานข้อมูล โดยส่งข้อมูลไปยัง Object addforum มีการเรียกใช้ ฟังก์ชัน add และทำการบันทึกข้อมูลลงตาราง foruminfo ในระบบฐานข้อมูล

บรรทัดที่ 39-42 เมื่อการตรวจสอบข้อมูลถูกต้อง ข้อมูลจะถูงส่งไฟยังฟังก์ชัน forums() ในไฟล์

เพื่อทำการบันทึกการตั้งกระทู้ และ ฟังก์ชัน forums() จะทำการส่งข้อมูลฟอร์มและผู้ใช้ไปยังหน้า View

บรรทัดที่ 43 คือ case เมื่อไม่พบสิทธิ์ผู้ใช้งาน จะส่งค่ากลับไปหน้าหลัก

สร้างไฟล์ Models ที่มีชื่อว่า Foruminfo.scala ขึ้นมาเพื่อใช้ในส่วนของการจัดการฐานข้อมูล และติดต่อกับส่วนการควบคุมการทำงาน ของ Controller

**ภาพที่ 4.8** หน้าหลักการตั้งกระทู้สนทนาในส่วน Models

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | case class Foruminfo (  id: String,  userID: String,  title: String,  detail: String,  picture: String  //date: Option[java.sql.Date]  )  class foruminfos(tag: Tag) extends Table[Foruminfo](tag, "foruminfo") {  def id = column[String]("id", O.PrimaryKey)  def userID = column[String]("userID")  **ภาพที่ 4.8** หน้าหลักการตั้งกระทู้สนทนาในส่วน Models  def title = column[String]("title")  def detail = column[String]("detail")  def picture = column[String]("picture")  // def date = column[String]("date")  def \* = ( id, userID, title, detail, picture ) <> (Foruminfo.tupled, Foruminfo.unapply)  }  object addforum {  val dbConfig = DatabaseConfigProvider.get[JdbcProfile](Play.current)  val dbforuminfo = TableQuery[foruminfos]  def add(role: Foruminfo): Future[String] = {  dbConfig.db.run(dbforuminfo += role).map(res => "successfully").recover {  case ex: Exception => ex.getCause.getMessage  }  }  } |

**ภาพที่ 4.8 (ต่อ)** หน้าหลักการตั้งกระทู้สนทนาในส่วน Models

จากภาพที่ 4.8 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-8 คือคลาส Artwork ที่เก็บค่าข้อมูลไว้เพื่อเชื่อมต่อกับฐานข้อมูล

บรรทัดที่ 9-17 คือการ Mapping ข้อมูลลงในตาราง foruminfo ในฐานข้อมูลของระบบ

บรรทัดที่ 20 คือการดึงข้อมูลจากตารางมาใช้

บรรทัดที่ 22-26 การเรียกใช้ฟังก์ชัน add เพื่อทำการเพิ่มข้อมูลลงฐานข้อมูล

1. การควบคุมการทำงานส่วนของฟังก์ชันการตั้งกระทู้สนทนา ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | POST /forum controllers.IndieApplication.forum |

**ภาพที่ 4.9** หน้าหลักการตั้งกระทู้สนทนาในส่วน Routes

จากภาพที่ 4.9 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ POST ผ่าน URL: /forum ซึ่งมีการเรียกใช้ฟังก์ชัน forum() โดย

Controllers มีชื่อไฟล์ว่า IndieApplication.scala

## 4.3 การพัฒนาในส่วนการเรียกดูข้อมูล

เมื่อสมาชิกต้องการเรียกดูข้อมูลจากระบบ โดยระบบจะทำการดึงข้อมูลที่สมาชิกต้องการเรียกดูออกมาแสดง ในส่วนนี้จะยกตัวอย่าง การดึงข้อมูลรายการโมเดล ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ IndieApplication.scala ArtWork.scala, modelUser.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การสร้างไฟล์ html โดยชื่อว่า createPosts.scala.html สำหรับการแสดงผล

**ภาพที่ 4.10** หน้าหลักการเรียกดูข้อมูลในส่วน View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @(user: models.User, dbartwork : Seq[models.ArtWork])(implicit messages: Messages)  @for(artwork  <- dbartwork ){  <div class="box">  <a class="image fit">  <img src="@routes.Assets.at("images/")@artwork.picture"alt=" " width="600"  height="200"/></a>  <div class="inner">  <h5>@artwork.title</h5>  <a href="@routes.IndieApplication.showmodel(artwork.userID )"  class="button fit" ><i class="fa fa-eye" aria-hidden="true"></i> Watch</a>  </div>  </div> } |

จากภาพที่ 4.10 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือ import models มาเพื่อเรียกใช้งานในหน้า html

บรรทัดที่ 2-11 คือการวนลูปเชคค่าข้อมูลในตาราง artwork ที่เก็บข้อมูลไว้ในฐานข้อมูล จากนั้นทำการ

get ข้อมูลในฐานข้อมูลออกมาแสดง

1. สร้างฟังก์ชัน model ในไฟล์ที่มีชื่อว่า IndieApplication.scala ขึ้นมาเพื่อใช้ในการเรียกดูรายการโมเดล และกำหนดการแสดงผลไปยัง View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def model = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  val data = for{  a <- uploadart.listAll  }yield(a)  data.map { dbartwork =>  Ok(views.html.modelsUser(user,dbartwork))  }  case None => Future.successful(Redirect(routes.ApplicationController.index()))  }  } |

**ภาพที่ 4.11** หน้าหลักการเรียกดูข้อมูลในส่วน Controllers

จากภาพที่ 4.11 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 ฟังก์ชัน model() คือการเรียกดูรายการโมเดลในระบบ และมีการตรวจสอบสิทธิ์ผู้ใช้

บรรทัดที่ 3-7 คือ การดึงข้อมูลในฐานข้อมูล โดยไปเรียกไฟล์ ArtWork.scala เพื่อเรียกใช้ Object

uploadart และฟังก์ชัน listAll ซึ่งเป็นการดึงข้อมูลมาทั้งหมด ในส่วน data.map คือการดึง ค่าจากตัวแปรไปใช้ และทำการส่งข้อมูลผู้ใช้ และข้อมูลโมเดลไปแสดงหน้า modelUser.scala.html

บรรทัดที่ 8 คือการตรวจสอบสิทธิ์ ซึ่งไม่พบบัญชีผู้ใช้ ระบบจะส่งไปหน้าหลัก

1. สร้างฟังก์ชันใน object uploadart ชื่อว่า listAll ของไฟล์ ArtWork เพื่อเรียกดูข้อมูล

|  |  |
| --- | --- |
| 1  2  3 | def listAll: Future[Seq[ArtWork]] = {  dbConfig.db.run(dbupload.result)  } |

**ภาพที่ 4.12** หน้าหลักการเรียกดูข้อมูลในส่วน Models

จากภาพที่ 4.12 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-3 ฟังก์ชัน listAll() คือการดึงข้อมูลจากฐานข้อมูล ตาราง artwork มาแสดงทั้งหมด

1. การควบคุมการทำงานส่วนของฟังก์ชันการตั้งกระทู้สนทนา ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | GET /model controllers.IndieApplication.model |

**ภาพที่ 4.13** หน้าหลักการเรียกดูข้อมูลในส่วน Routes

จากภาพที่ 4.13 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ GET ผ่าน URL: /model ซึ่งมีการเรียกใช้ฟังก์ชัน model() โดย

Controllers มีชื่อไฟล์ว่า IndieApplication.scala

## 4.4 การพัฒนาในส่วนการแก้ไขข้อมูล

เมื่อสมาชิกต้องการแก้ไขข้อมูลในระบบ โดยระบบจะทำการดึงข้อมูลเดิมมาแสดง เพื่อทำการแก้ไข และทำการบันทึกข้อมูลที่ทำการแก้ไขลงในระบบ ในส่วนนี้จะยกตัวอย่าง การแก้ไขกระทู้สนทนา ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ ApplicationController.scala, Foruminfo.scala, EditPost.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การสร้างไฟล์ html โดยชื่อว่า postsUser.scala.html สำหรับการแสดงผล

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @import b3.inline.fieldConstructor  <div class="col-sm-8 “>  @helper.form(action = routes.IndieApplication.forum,'enctype -> "multipart/form-data") {  @helper.CSRF.formField  <h5 >หัวข้อกระทู้</h5>  @b3.text( forumForm("title"), '\_label -> "Topic" )  @b3.textarea( forumForm("detail"), '\_label -> "details" ,'rows -> 3 )  @b3.file( forumForm("picture"), '\_label -> "File" )  <div class=" text-right">  <button type="submit" class="btn btn-primary">Submit</button> &nbsp; &nbsp;  <button type="submit" class="btn btn-danger">Cancel</button>  </div>  } |

ภาพที่ 4.14 หน้าหลักของการแก้ไขกระทู้สนทนาในส่วน View

จากภาพที่ 4.14 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 6 คือช่องสำหรับกรอกข้อมูลหัวข้อกระทู้

บรรทัดที่ 7 คือช่องสำหรับกรอกรายละเอียดกระทู้

บรรทัดที่ 8 คือช่องสำหรับอัพโหลดไฟล์รูปภาพ

บรรทัดที่ 10 คือปุ่มสำหรับส่งข้อมูลการตั้งกระทู้ซึ่งจะเรียกใช้ฟังก์ชัน updatePost2 ในไฟล์

ApplicationController.scala

บรรทัดที่ 11 คือปุ่มสำหรับยกเลิกการตั้งกระทู้ และกลับสู่หน้าหลักของระบบ

1. สร้างฟังก์ชัน updatePost2() ในไฟล์ที่มีชื่อว่า ApplicationController.scala ขึ้นมาเพื่อใช้ในการแก้ไข และกำหนดการแสดงผลไปยัง View

**ภาพที่ 4.15** หน้าหลักของการแก้ไขกระทู้สนทนาในส่วน Controller

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | def updatePost2= UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  request.body.asMultipartFormData.map {a =>  val datatitle = a.dataParts.get("title").map { a =>  for{  b <- a.mkString("")  }yield b  }  val datadetail = a.dataParts.get("detail").map { a =>  for{  b <- a.mkString("")  }yield b  }  val dataimg = a.file("picture").map { a=>  val filename = a.filename  val extension = FilenameUtils.getExtension(filename)  val newFileName = s"${UUID.randomUUID}.$extension"  a.ref.moveTo(new File(s"public/images/$newFileName"))  for{  b <- newFileName  }yield b  }  val getRelation = for{  a <- addforum.get(uid)  }yield a  val title = getdata(datatitle)  val detail = getdata(datadetail)  val picture = getdata(dataimg)  //add table database  val aaaa = getRelation.map { data =>  data.map { aa =>  val dbforum = Foruminfo (  id = aa.id,  userID = user.userID.toString,  title = title ,  detail = detail ,  picture = picture  )  val saveDate = for{  a <- addforum.update(dbforum)  }yield a  }  }  Future.successful(Redirect("/posts"))  }.getOrElse {  Future.successful(Redirect("/fourums"))  }  case None => Future.successful(Redirect("/"))  }  }  **ภาพที่ 4.15 (ต่อ)** หน้าหลักของการแก้ไขกระทู้สนทนาในส่วน Controller |

จากภาพที่ 4.15 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-2 ฟังก์ชัน forum() คือการตั้งกระทู้สนทนาของสมาชิกในระบบ โดยจะมีการตรวจสอบสิทธิ์

บรรทัดที่ 3-13 คือ การดึงข้อมูลจาก View มาเป็น Object ที่เป็น List จากนั้นนำข้อมูลมาแปลงจาก

Object List เป็น Object และทำการแปลงข้อมูลจาก Object เป็น String เพื่อทำการ บันทึกข้อมูลลงฐานข้อมูลของระบบ

บรรทัดที่ 14-22 คือ การเพิ่มไฟล์ โดยมีการกำหนดพาทสำหรับเก็บข้อมูลของไฟล์รูปภาพ ไปยังระบบ

และมีการสุ่มชื่อไฟล์ให้ใหม่ทุกครั้งที่ทำการอัพโหลดรูปภาพ

บรรทัดที่ 23-25 ฟังก์ชัน getRelation คือการ get userID ของผู้ใช้ที่ทำการแก้ไข

บรรทัดที่ 26-28 คือ การเรียกใช้ ฟังก์ชัน getdata เพื่อแปลง Option ให้เป็น String

บรรทัดที่ 29-38 ฟังก์ชัน aa ทำการดึงข้อมูล โดยส่งไปยัง dbforum คือการส่งข้อมูลไปยังไฟล์

Foruminfo.scala เพื่อทำการตรวจสอบข้อมูล และบันทึกลงในฐานข้อมูล

บรรทัดที่ 39-43 ฟังก์ชัน saveDate คือการเพิ่มข้อมูลการตั้งกระทู้ที่แก้ไขใหม่ลงในฐานข้อมูล โดยส่ง

ข้อมูลไปยัง Object addforum ในไฟล์ Foruminfo.scala โดยเรียกใช้ ฟังก์ชัน update และทำการบันทึกข้อมูลลงตาราง foruminfo ในระบบฐานข้อมูล

บรรทัดที่ 44-47 เมื่อการตรวจสอบข้อมูลถูกต้อง ข้อมูลจะถูงส่งไฟยังฟังก์ชัน forums() ในไฟล์

เพื่อทำการบันทึกการตั้งกระทู้ และ ฟังก์ชัน posts() จะทำการส่งข้อมูลกระทู้และผู้ใช้ไปยังหน้า View

บรรทัดที่ 48 คือ case เมื่อไม่พบสิทธิ์ผู้ใช้งาน จะส่งค่ากลับไปหน้าหลัก

1. สร้างฟังก์ชันใน object addforum ชื่อว่า update ของไฟล์ Foruminfo เพื่อแก้ไขข้อมูลกระทู้

|  |  |
| --- | --- |
| 1  2  3  4  5 | def update(foruminfo: Foruminfo): Future[String] = {  dbConfig.db.run(dbforuminfo.filter(\_.id ===  foruminfo.id).update(foruminfo)).map(res => "successfully").recover {  case ex: Exception => ex.getCause.getMessage  }  } |

**ภาพที่ 4.16** หน้าหลักของการแก้ไขกระทู้สนทนาในส่วน Models

จากภาพที่ 4.16 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-5 ฟังก์ชัน update() คือการดึงข้อมูลจากฐานข้อมูลมาเปรียบเทียบ และบันทึกข้อมูลล่าสุดแทนที่ข้อมูลเดิม ลงตาราง foruminfo

1. การควบคุมการทำงานส่วนของฟังก์ชันการแก้ไขกระทู้สนทนา ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | POST /updatePost2 controllers.ApplicationController.updatePost2 |

**ภาพที่ 4.17** หน้าหลักการเรียกดูข้อมูลในส่วน Routes

จากภาพที่ 4.17 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ POST ผ่าน URL: / updatePost2 ซึ่งมีการเรียกใช้ฟังก์ชัน

updatePost2() โดย Controllers มีชื่อไฟล์ว่า ApplicationController.scala

## 4.5 การพัฒนาในส่วนการลบข้อมูล

เมื่อสมาชิกต้องการข้อมูลในระบบ และทำการบันทึกข้อมูลที่ทำการลบลงในระบบ ในส่วนนี้จะยกตัวอย่าง การลบคว่ามคิดเห็น ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ ApplicationController.scala, Comment.scala, showposta.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การแก้ไขไฟล์ html โดยเพิ่มส่วนของการลบความคิดเห็น ในไฟล์ showposta.scala.html สำหรับการแสดงผล

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <div class="panel-heading">  <div class=" text-right">  <a href="# "><i class="fa fa-pencil-square-o" aria-hidden="true"></i></a>  &nbsp;  <a data-toggle="modal" href="#modeldelete2"><i class="fa fa-trash-o"  aria-hidden="true"></i></a>  </div>  </div> |

**ภาพที่ 4.18** หน้าหลักการลบข้อมูลความคิดเห็นในส่วน Routes

จากภาพที่ 4.18 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 4 คือปุ่มสำหรับการลบข้อมูลความคิดเห็น ซึ่งมีการเรียกใช้ฟังก์ชัน deleteComment(id) ใน

ไฟล์มีชื่อว่า ApplicationController.scala

1. สร้างฟังก์ชัน deleteComment() ในไฟล์ที่มีชื่อว่า ApplicationController.scala ขึ้นมาเพื่อใช้ในการแก้ไข และกำหนดการแสดงผลไปยัง View

**ภาพที่ 4.19** หน้าหลักการลบข้อมูลความคิดเห็นในส่วน Controllers

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | def deleteComment (id : String) = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  val c = for{  a <- addcomment.delete(id)  }yield a  Future.successful(Redirect("/"))  case None => Future.successful(Redirect("/"))  }  } |

จากภาพที่ 4.19 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 ฟังก์ชัน deleteComment(id) จะส่ง id ที่ต้องการลบ เพื่อนำไปค้นหาข้อมูลที่ต้องการลบ

โดยจะส่งไป Object addcomment และเรียกใช้ฟังก์ชัน delete(id) เพื่อทำการลบข้อมูล

ความ คิดเห็นที่ต้องการลบในระบบ

1. สร้างฟังก์ชันใน object addcomment ชื่อว่า delete(id) ของไฟล์ Comment เพื่อลบความคิดเห็น

|  |  |
| --- | --- |
| 1  2  3 | def delete(userID: String): Future[Int] = {  dbConfig.db.run(dbcomment.filter(\_.userID=== userID).delete)  } |

**ภาพที่ 4.20** หน้าหลักการลบข้อมูลความคิดเห็นในส่วน Models

จากภาพที่ 4.18 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 4 ฟังก์ชัน delete(id) คือการนำ id ที่รับมา เพื่อไปค้นหาข้อมูลในตาราง จากนั้นทำการลบข้อมูล

1. การควบคุมการทำงานส่วนของฟังก์ชันการลบความคิดเห็น ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | GET /deleteComment/:id controllers.ApplicationController.deleteComment(id : String) |

**ภาพที่ 4.21** หน้าหลักการลบความคิดเห็นในส่วน Routes

จากภาพที่ 4.17 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ GET ผ่าน URL: / deleteComment/:id โดยมีการส่ง id ไปค้นหา

ข้อมูลที่ต้องการลบ ซึ่งมีการเรียกใช้ฟังก์ชัน deleteComment() โดย Controllers มีชื่อไฟล์ว่า ApplicationController.scala

## 4.6 การพัฒนาในส่วนการแสดงความคิดเห็น

เมื่อสมาชิกต้องการแสดงความคิดเห็นในระบบ และทำการบันทึกข้อมูลที่ทำการแสดงความคิดเห็นลงในระบบ ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ IndieApplication.scala, Comment.scala, showposta.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การเพิ่มส่วนแสดงความคิดเห็นในไฟล์ html ในไฟล์ชื่อว่า showposta.scala.html สำหรับการแสดงผล

**ภาพที่ 4.22** หน้าหลักการแสดงความคิดเห็นในส่วน View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @import b3.inline.fieldConstructor  <div class="col-sm-10">  @helper.form(action = routes.IndieApplication.comment()) {  @helper.CSRF.formField  @b3.hidden( "commenterID", user.userID )  @b3.textarea( Forms("detail"), '\_label -> "Comment" ,'rows -> 3 )  <div class=" text-right" >  <button type="submit" class="btn btn-info ">send</button>  </div>  }  </div> |

จากภาพที่ 4.22 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 6 คือช่องสำหรับกรอกข้อมูลรายละเอียดความคิดเห็น

บรรทัดที่ 8 คือปุ่มสำหรับส่งข้อมูลการแสดงความคิดเห็น

1. สร้างฟังก์ชัน comment() ในไฟล์ที่มีชื่อว่า IndieApplication.scala ขึ้นมาเพื่อใช้ในการแสดงคิดเห็นและกำหนดการแสดงผลไปยัง View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | def comment = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  CommentForm.form.bindFromRequest.fold(  form => Future.successful( Redirect("/")),  data => {  val comments = Comment(  id = UUID.randomUUID.toString,  userID = data.commenterID,  detail = Some(data.detail),  artworkid = artworkid,  forumid = forumid  )  val saveD = for {  add <- addcomment.add(comments)  }yield add  Future.successful(Redirect("/"))  } )  case None => Future.successful(Redirect("/"))  }  } |

**ภาพที่ 4.23(ต่อ)** หน้าหลักการลบข้อมูลความคิดเห็นในส่วน Controllers

จากภาพที่ 4.23 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 ฟังก์ชัน deleteComment(id) จะส่ง id ที่ต้องการลบ เพื่อนำไปค้นหาข้อมูลที่ต้องการลบ

โดยจะส่งไป Object addcomment และเรียกใช้ฟังก์ชัน delete(id) เพื่อทำการลบข้อมูลความคิดเห็นที่ต้องการลบในระบบ

1. สร้างไฟล์ Models ที่มีชื่อว่า Comment.scala ขึ้นมาเพื่อใช้ในส่วนของการจัดการฐานข้อมูล และติดต่อกับส่วนการควบคุมการทำงาน ของ Controller

**ภาพที่ 4.24** หน้าหลักการลบข้อมูลความคิดเห็นในส่วน Models

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | case class Comment (  id: String,  userID: String,  detail: Option[String],  artworkid: String,  forumid: String  )  class comments (tag: Tag) extends Table[Comment](tag, "comment") {  def id = column[String]("id", O.PrimaryKey)  def userID = column[String]("userID")  def detail = column[Option[String]]("detail")  def artworkid = column[String]("artworkid")  def forumid = column[String]("forumid")  def \* = (id, userID, detail, artworkid, forumid) <> (Comment.tupled,  Comment.unapply)  }  object addcomment {  val dbConfig = DatabaseConfigProvider.get[JdbcProfile](Play.current)  val dbcomment = TableQuery[comments]  def add(comment : Comment): Future[String] = {  dbConfig.db.run(dbcomment += comment).map(res => "successfully").recover {  case ex: Exception => ex.getCause.getMessage  }  } |

จากภาพที่ 4.24 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-8 คือคลาส Comment ที่เก็บค่าข้อมูลไว้เพื่อเชื่อมต่อกับฐานข้อมูล

บรรทัดที่ 9-16 คือการ Mapping ข้อมูลลงในตาราง Comment ในฐานข้อมูลของระบบ

บรรทัดที่ 17-24 Object addcomment มีไว้เพื่อเก็บฟังก์ชัน โดยมีการเรียกใช้ฟังก์ชัน add จากไฟล์

IndieApplication.scala ในฟังก์ชัน comment เพื่อทำการเพิ่มข้อมูลลงฐานข้อมูล

1. การควบคุมการทำงานส่วนของฟังก์ชันการแสดงความคิดเห็น ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | POST /comment controllers.IndieApplication.comment |

**ภาพที่ 4.25** หน้าหลักการแสดงความคิดเห็นในส่วน Routes

จากภาพที่ 4.25 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ POST ผ่าน URL: / Comment ซึ่งมีการเรียกใช้ฟังก์ชัน

Comment() โดย Controllers ที่มีชื่อไฟล์ว่า IndieApplication.scala

## 4.7 การพัฒนาในส่วนการติดตามสมาชิก

เมื่อสมาชิกต้องการติดตามผู้ใช้ในระบบ เพื่อติดตามผลงานของสมาชิกที่ทำการติดตาม ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ IndieApplication.scala, Follow.scala, member.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

1. การสร้างไฟล์ html โดยชื่อว่า member.scala.html สำหรับการแสดงผล

**ภาพที่ 4.26** หน้าหลักการติดตามสมาชิกในส่วน View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @for (u  <- users){  @if(u.userID.toString != user.userID.toString){  <div class=" col-sm-2">  <div class="box2">  <a class="image fit"><img src="@routes.Assets.at("images/user2.jpg")"  width="300" height="150" /></a>  <div class="inner">  <a href="@routes.IndieApplication.profile(u.userID)"><h5>@u.fullName</h5></a>  @for(f <- dbfollow){  @if(f.fid == u.userID.toString && user.userID.toString == f.userID){  <a href="#" > <button class="button fit btn-sm"><i class="fa fa-plus">  </i> UnFollow</button>  </a>  } } |

**ภาพที่ 4.26(ต่อ)** หน้าหลักการติดตามสมาชิกในส่วน View

จากภาพที่ 4.26 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-2 เป็นการวนลูป array เพื่อนำข้อมูลผู้ใช้มาใช้งาน

บรรทัดที่ 3 กำหนดเงื่อนไข userID ไม่เท่ากับ userID ของผู้ใช้งาน เพื่อแสดงข้อมูลสมาชิกท่านอื่น

บรรทัดที่ 9 เป็นปุ่มสำหรับ Follow สมาชิก เมื่อกดปุ่มจะส่ง userID ของผู้ที่เลือกติดตาม

บรรทัดที่ 10 เป็นการวนลูป array เพื่อนำข้อมูลการติดตามมาใช้งาน

บรรทัดที่ 11 กำหนดเงื่อนไข fid เท่ากับ userID ของผู้ใช้งาน ซึ่ง fid คือรหัสสมาชิกที่ทำการติดตามแล้ว

บรรทัดที่ 12 เป็นปุ่มสำหรับ UnFollow สมาชิกที่ติมตามแล้ว

1. สร้างฟังก์ชัน addfollow() ในไฟล์ที่มีชื่อว่า IndieApplication.scala ขึ้นมาเพื่อใช้ในการติดตามสมาชิกและกำหนดการแสดงผลไปยัง View

**ภาพที่ 4.27** หน้าหลักการติดตามสมาชิกในส่วน Controllers

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | def addfollow (fid :String) = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  val a = Follow(  id = Some(0) ,  fid = fid ,  userID = user.userID.toString  )  val n = for{  c <- DBfollow.add(a)  }yield c  Future.successful(Redirect("/member"))  case None => Future.successful(Redirect("/"))  }  } |

**ภาพที่ 4.27(ต่อ)** หน้าหลักการติดตามสมาชิกในส่วน Controllers

จากภาพที่ 4.27 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 ฟังก์ชัน addfollow(id) คือเมื่อมีการเรียกใช้ ระบบจะทำการส่ง id ของผลงานผู้ติดตาม

ที่เลือกไปบันทึกในฐานข้อมูล

บรรทัดที่ 4 กำหนด val a เท่ากับ คลาส Follow

บรรทัดที่ 5 กำหนด id มีค่าเริ่มจาก 0 เมื่อมีการเรียกใช้ครั้งถัดไปค่าจะเพิ่มขึ้นเรื่อยๆ

บรรทัดที่ 6 กำหนด fid เท่ากับ fid ของผู้ต้องการติดตามที่ผู้ใช้งานเลือก

บรรทัดที่ 7 กำหนด userID เท่ากับ userID ที่เข้าใช้งานระบบ

บรรทัดที่ 9-11 กำหนดให้ n วนค่าข้อมูลที่เป็น Object DBfollow เพื่อทำการ add ข้อมูลลงใน

ฐานของมูลชองระบบ

บรรทัดที่ 12 เมื่อการทำงานถูกต้องระบบจะส่งไปหน้า View

บรรทัดที่ 13 เมื่อการทำงานไม่ถูกต้องระบบจะส่งไปหน้าแรกของสมาชิก

1. สร้างไฟล์ Models ที่มีชื่อว่า Follow.scala ขึ้นมาเพื่อใช้ในส่วนของการจัดการฐานข้อมูล และติดต่อกับส่วนการควบคุมการทำงาน ของ Controller

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | case class Follow (  id: Option[Long],  fid : String,  userID : String  )  class follows (tag: Tag) extends Table[Follow](tag, "follow") {  def id = column[Long]("id", O.PrimaryKey, O.AutoInc)  def fid = column[String]("fid")  def userID = column[String]("userID")  override def \* = (id.? ,fid, userID ) <> (Follow.tupled, Follow.unapply)  }  object DBfollow {  val dbConfig = DatabaseConfigProvider.get[JdbcProfile](Play.current)  val dbfollow = TableQuery[follows]  def add(follow: Follow): Future[String] = {  dbConfig.db.run(dbfollow += follow).map(res => "successfully").recover {  case ex: Exception => ex.getCause.getMessage  }  } |

**ภาพที่ 4.28** หน้าหลักการติดตามสมาชิกในส่วน Models

จากภาพที่ 4.28 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-5 คือคลาส Follow ที่เก็บค่าข้อมูลไว้เพื่อเชื่อมต่อกับฐานข้อมูล

บรรทัดที่ 7-12 คือการ Mapping ข้อมูลลงในตาราง follow ในฐานข้อมูลของระบบ

บรรทัดที่ 13-20 Object DBfollow มีไว้เพื่อเก็บฟังก์ชัน โดยมีการเรียกใช้ฟังก์ชัน add จากไฟล์

IndieApplication.scala ในฟังก์ชัน addfollow() เพื่อทำการเพิ่มข้อมูลลงฐานข้อมูล

1. การควบคุมการทำงานส่วนของฟังก์ชันการติดตามสมาชิก ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | GET /addfollow/:fid controllers.IndieApplication.addfollow(fid : String) |

**ภาพที่ 4.29** หน้าหลักการแสดงความคิดเห็นในส่วน Routes

จากภาพที่ 4.29 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ GET ผ่าน URL: / addfollow/:fid

โดยมีการส่ง id ไปค้นหาข้อมูลที่ต้องการติดตาม ซึ่งมีการเรียกใช้ฟังก์ชัน addfollow() โดย Controllers มีชื่อไฟล์ว่า IndieApplication.scala

## 4.8 การพัฒนาในส่วนแฟ้มสะสมผลงาน

สมาชิกเพิ่มผลงานโมเดลสามมิติเข้าแฟ้มสะสมผลงาน ซึ่งใช้ไฟล์ที่เกี่ยวข้อง 4 ไฟล์ ได้แก่ IndieApplication.scala, Collection.scala, collection.scala.html และ Routes ซึ่งมีขั้นตอนในการอัพโหลดดังนี้

* 1. การสร้างไฟล์ html โดยชื่อว่า collection.scala.html สำหรับการแสดงผล

**ภาพที่ 4.30** หน้าหลักการแฟ้มสะสมผลงานในส่วน View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | @(user: models.User, col : Seq[models.Collection],dbartwork : Seq[models.ArtWork])(implicit messages: Messages)  @for(artwork  <- dbartwork ){  @for(c <- col){  @if(artwork.id == c.artworkid){  <div class=" col-sm-3">  <div class="box2">  <a class="image fit"><img  src="@routes.Assets.at("images/")@artwork.picture"alt=""  width="300" height="150"/></a>  <div class="inner">  <h5>@artwork.title</h5>  <a href="@routes.IndieApplication.showmodel(artwork.id, artwork.userID )"  class="button btn-sm" ><i class="fa fa-eye" aria-hidden="true"></i>  Watch</a>  <a href="#" class="button btn-sm" ><i class="fa fa-trash-o" aria-  hidden="true"></i> Delete</a>  </div>  </div>  </div><!--col-sm3-->  }  }  } |

**ภาพที่ 4.30(ต่อ)** หน้าหลักการแฟ้มสะสมผลงานในส่วน View

จากภาพที่ 4.30 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการ import ตาราง เพื่อเรียกใช้ข้อมูล

บรรทัดที่ 2-3 คือการวดลูปเพื่อค้นหาข้อมูลในตาราง artwork เพื่อดึงข้อมูลมาแสดง

บรรทัดที่ 4 คือการวดลูปเพื่อค้นหาข้อมูลในตาราง collection มาใช้งาน

บรรทัดที่ 5 คือการเช็คเงื่อนไข โดยที่กำหนดให้ id ในตาราง artwork มีค่าเท่ากับ id ในตาราง

collection เมื่อข้อมูลตรงกันให้ทำการดึงข้อมูลผลงานมาแสดง

บรรทัดที่ 8 คือการดึงข้อมูลรูปภาพในตาราง artwork มาแสดง

บรรทัดที่ 10 คือการดึงข้อมูลหัวข้อผลงานในตาราง artwork มาแสดง

บรรทัดที่ 11 เมื่อกดปุ่ม ดูโมเดล ระบบจะส่งค่า id และ userID พร้อมแสดงหน้ารายละเอียดผลงาน

บรรทัดที่ 12 เมื่อทำการกดปุ่ม Delete ระบบจะทำการลบผลงานออกจากแฟ้มสะสมผลงาน

* 1. สร้างฟังก์ชัน addcollection() ในไฟล์ที่มีชื่อว่า IndieApplication.scala ขึ้นมาเพื่อใช้เพิ่มผลงานเข้าแฟ้มสะสมผลงาน และกำหนดการแสดงผลไปยัง View

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | def addcollection (id :String ) = UserAwareAction.async { implicit request =>  request.identity match {  case Some(user) =>  val a = Collection(  id = Some(0) ,  userID = user.userID.toString,  artworkid = id  )  val n = for{  c <- DBcollection.add(a)  }yield c  Future.successful(Redirect("/col"))  case None => Future.successful(Redirect("/"))  } } |

**ภาพที่ 4.31** หน้าหลักการแฟ้มสะสมผลงานในส่วน Controllers

จากภาพที่ 4.31 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 ฟังก์ชัน addfollow(id) คือเมื่อมีการเรียกใช้ ระบบจะทำการส่ง id ของผลงานโมเดลสาม

มิติที่เลือกไปบันทึกในฐานข้อมูล

บรรทัดที่ 3 กำหนด a เท่ากับ คลาส Collection

บรรทัดที่ 4 กำหนด id มีค่าเริ่มจาก 0 เมื่อมีการเรียกใช้ครั้งถัดไปค่าจะเพิ่มขึ้นเรื่อยๆ

บรรทัดที่ 5 กำหนด userID เท่ากับ userID ที่เข้าใช้งานระบบ

บรรทัดที่ 6 กำหนด artworkid เท่ากับ id ของผลงานโมเดลสามมิติที่ผู้ใช้เลือก

บรรทัดที่ 8-10 ดึงค่าข้อมูลที่เป็น Object DBcollection เพื่อทำการ add ข้อมูลลงใน

ฐานของมูลชองระบบ

บรรทัดที่ 11 เมื่อการทำงานถูกต้องระบบจะส่งไปหน้า View

บรรทัดที่ 12 เมื่อการทำงานไม่ถูกต้องระบบจะส่งไปหน้าแรกของสมาชิก

* 1. สร้างไฟล์ Models ที่มีชื่อว่า Collection.scala ขึ้นมาเพื่อใช้ในส่วนของการจัดการฐานข้อมูล และติดต่อกับส่วนการควบคุมการทำงาน ของ Controller

**ภาพที่ 4.32** หน้าหลักการแฟ้มสะสมผลงานในส่วน Models

**ภาพที่ 4.** หน้าหลักการแฟ้มสะสมผลงานในส่วน Models

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | case class Collection(  id: Option[Long],  artworkid: String,  userID: String  )  class collections(tag: Tag) extends Table[Collection](tag, "collection") {  def id = column[Long]("id", O.PrimaryKey, O.AutoInc)  def artworkid = column[String]("artworkid")  def userID = column[String]("userID")  override def \* = (id.? , artworkid, userID ) <> (Collection.tupled, Collection.unapply)  }  object DBcollection {  val dbConfig = DatabaseConfigProvider.get[JdbcProfile](Play.current)  val dbcol = TableQuery[collections]  def add(collection: Collection): Future[String] = {  dbConfig.db.run(dbcol += collection).map(res => "successfully").recover {  case ex: Exception => ex.getCause.getMessage  }  } |

**ภาพที่ 4.32** หน้าหลักการแฟ้มสะสมผลงานในส่วน Models

จากภาพที่ 4.32 สามารถอธิบายรายละเอียดการทำงานได้ดังนี้

บรรทัดที่ 1-5 คือคลาส Collection ที่เก็บค่าข้อมูลไว้เพื่อเชื่อมต่อกับฐานข้อมูล

ประกอบไปด้วย id คือลำดับของข้อมูล userID คือรหัสของผู้ใช้งาน และ artworkid คือรหัส

ของผลงานสามมิติ

บรรทัดที่ 6-11 คือการ Mapping ข้อมูลลงในตาราง collection ในฐานข้อมูลของระบบ

บรรทัดที่ 12-19 Object DBcollection มีไว้เพื่อเก็บฟังก์ชัน โดยมีการเรียกใช้ฟังก์ชัน add จากไฟล์

IndieApplication.scala ในฟังก์ชัน addcollection() เพื่อทำการเพิ่มข้อมูลลงฐานข้อมูล

* 1. การควบคุมการทำงานส่วนของฟังก์ชันเพิ่มแฟ้มสะสมผลงาน ผ่านไฟล์ Routes

|  |  |
| --- | --- |
| 1 | GET /addcollection/:id controllers.IndieApplication.addcollection(id : String) |

**ภาพที่ 4.33** หน้าหลักการแสดงความคิดเห็นในส่วน Routes

จากภาพที่ 4.33 สามารถอธิบายรายะเอียดการทำงานได้ดังนี้

บรรทัดที่ 1 คือการส่งค่าข้อมูลแบบ GET ผ่าน URL: addcollection/:id

โดยมีการส่ง id ไปค้นหาข้อมูลที่ต้องการเพิ่มเข้าแฟ้มสะสมผลงาน ซึ่งมีการเรียกใช้ฟังก์ชัน addcollection() โดย Controllers มีชื่อไฟล์ว่า IndieApplication.scala