MIDTERM2

Team members : Nursultan Myrzagulov 200103440
Zhandos Saparbayev 200103201
Zhiger Sairan 200113005

1 Filtering items based on price, rating

```go
// filtering the books by arrange two price

func PriceFiltering(c *gin.Context) {
	from := c.Query("from") // getting first price
	to := c.Query("to")     // getting second price
	var books []models.Book
	// get all books from table with price between this two prices
	result := initializers.GetDB().Where("price >= ?", from).Where("price <= ?", to).Order("price").Find(&books)
	// return the error if failed to get books between two price
	if result.Error != nil {
		c.JSON(http.StatusBadRequest, gin.H{
			"Error": "failed to get books between of prices",
		})
		return
	}

	c.JSON(http.StatusOK, gin.H{
		"result": books,
	})
}

// filtering books by rating

func RatingFiltering(c *gin.Context) {
	rating := c.Query("rating") // getting book rating
	var books []models.Book     // taking all books with rating >= target rating
	result := initializers.GetDB().Where("rating >= ?", rating).Order("rating desc").Find(&books)

	if result.Error != nil {
		c.JSON(http.StatusBadRequest, gin.H{
			"Error": "failed to get books by rating",
		})
		return
	}

	c.JSON(http.StatusOK, gin.H{
		"result": books,
	})
}
```

2 Giving rating for items (rating can only be given by the client)

```go
// for giving a ratings by clients

func GiveRating(c *gin.Context) {
	var body struct { // taking book title and rating for the book
		Title  string //target title
		Rating float32
	}

	if c.Bind(&body) != nil {
		c.JSON(http.StatusBadRequest, gin.H{
			"error": "Failed to read body",
		})
		return
	}

	var target models.Book
	initializers.GetDB().Find(&target, "title=?", body.Title) // taking book with target title
	avg := (body.Rating + target.Rating) / 2
	if body.Rating <= 5.0 { // if giving rating less or equal to 5 point then we update the rating for this book
		initializers.GetDB().Model(&target).Update("Rating", avg)
	}
	c.JSON(http.StatusOK, gin.H{
		"Change book rating": avg,
	})
}
```

3 Commenting items

```
// for removing the comment by id

func DeleteComment(c *gin.Context) {
        id := c.Param("id") // get target comment id
        var targetComment models.Comment
        initializers.GetDB().Find(&targetComment, "id=?", id) // getting comment with by id
        var admin models.User
        initializers.GetDB().Find(&admin, "email=?", GetUserEmail(c)) // getting user who want to delete the comment

        if GetUserEmail(c) != targetComment.Author && admin.Type != "Admin" { // checking for owners the comment
                c.JSON(http.StatusBadRequest, gin.H{
                        "Error": "Only owners can delete his comments",
                })
                return
        }

        initializers.GetDB().Delete(&targetComment, "id=?", id) // if its comment owner then we delete it
        c.JSON(http.StatusOK, gin.H{
                "Comment": "Successfully removed",
        })
}

// for getting all comments from table

func GetAllComments(c *gin.Context) {
        var comments []models.Comment
        initializers.GetDB().Find(&comments)
        c.JSON(http.StatusOK, gin.H{
                "Comments": comments,
        })
}
```


```
// for removing the comment by id

func DeleteComment(c *gin.Context) {
        id := c.Param("id") // get target comment id
        var targetComment models.Comment
        initializers.GetDB().Find(&targetComment, "id=?", id) // getting comment with by id
        var admin models.User
        initializers.GetDB().Find(&admin, "email=?", GetUserEmail(c)) // getting user who want to delete the comment

        if GetUserEmail(c) != targetComment.Author && admin.Type != "Admin" { // checking for owners the comment
                c.JSON(http.StatusBadRequest, gin.H{
                        "Error": "Only owners can delete his comments",
                })
                return
        }

        initializers.GetDB().Delete(&targetComment, "id=?", id) // if its comment owner then we delete it
        c.JSON(http.StatusOK, gin.H{
                "Comment": "Successfully removed",
        })
}

// for getting all comments from table

func GetAllComments(c *gin.Context) {
        var comments []models.Comment
        initializers.GetDB().Find(&comments)
        c.JSON(http.StatusOK, gin.H{
                "Comments": comments,
        })
}
```

```go
// for getting target comments for books (by id)

func GetCommentsByID(c *gin.Context) {
        id := c.Param("id") // get target id
        var target models.Book
        initializers.GetDB().Find(&target, "id=?", id) // take target book
        var comments []models.Comment
        initializers.GetDB().Find(&comments, "book=?", target.Title) // get all comment for this book
        c.JSON(http.StatusOK, gin.H{
                "Comments": comments,
        })
}

// for getting target comments for books (by book titles)

func GetCommentsForBook(title string) []models.Comment {
        var comments []models.Comment
        initializers.GetDB().Find(&comments, "book=?", title)
        return comments
}
```