Репорт на мидку 2. Тут все рекуайрменты:
1) Filter:

```
func ProductList(c *fiber.Ctx) error {
	//Token authenticate
	headerToken := c.Get("Authorization")
	if headerToken == "" {
		return c.Status(401).JSON(fiber.Map{
			"success": false,
			"message": "Unauthorized",
			"error":   map[string]interface{}{},
		})
	}
	if err := middleware.AuthenticateToken(middleware.SplitToken(headerToken)); err != nil {
		return c.Status(401).JSON(fiber.Map{
			"success": false,
			"message": "Unauthorized",
			"error":   map[string]interface{}{},
		})
	}
	//Token authenticate
	limit := c.Query("limit")
	skip := c.Query("skip")
	categoryId := c.Query("categoryId")
	productName := c.Query("name")
	price := c.Query("price")
	intLimit, _ := strconv.Atoi(limit)
	intSkip, _ := strconv.Atoi(skip)
	var products []models.Product
	var ratings []models.Rating
	productsRes := make([]*models.ProductResult, 0)
	if productName == "" {
		//filter by price
		var count int64
		var averageRating float64
		db.DB.Where("price = ?",
price).Limit(intLimit).Offset(intSkip).Find(&products).Count(&count)
		var category models.Category
```

```
                var discount models.Discount
                var rating models.Rating
                for i := 0; i < len(products); i++ {
                        db.DB.Table("categories").Where("id = ?",
products[i].CategoryId).Find(&category)
                        db.DB.Where("id = ?",
products[i].DiscountId).Limit(intLimit).Offset(intSkip).Find(&discount).Count(&count)
                        db.DB.Find(&ratings, "product_rating")
                        db.DB.Model(&rating).Select("AVG(product_rating)").Where("product_id = ?",
products[i].Id).Scan(&averageRating)
                        count = int64(len(products))
                        //productsRes =
                        productsRes = append(productsRes,
                                &models.ProductResult{
                                        Id:            products[i].Id,
                                        Sku:           products[i].Sku,
                                        Name:          products[i].Name,
                                        Stock:         products[i].Stock,
                                        Price:         products[i].Price,
                                        Image:         products[i].Image,
                                        Category:      category,
                                        Discount:      discount,
                                        ProductRating: averageRating,
                                        Rating:        rating,
                                },
                        )
                }
                meta := map[string]interface{}{
                        "total":  count,
                        "limit":  limit,
                        "skip":   skip,
                        "Rating": averageRating,
                }
                return c.Status(200).JSON(fiber.Map{
                        "success": true,
                        "message": "Success",
                        "data": map[string]interface{}{
```

```
                        "products": productsRes,
                        "meta":     meta,
                    },
                })
                    } else {
                var count int64
                var averageRating float64
                if categoryId != "" {
                    db.DB.Where("category_id = ?",
categoryId).Limit(intLimit).Offset(intSkip).Find(&products).Count(&count)
                } else {
                    db.DB.Where(" name= ?",
productName).Limit(intLimit).Offset(intSkip).Find(&products).Count(&count)
                }
                var category models.Category
                var discount models.Discount
                var rating models.Rating
                for i := 0; i < len(products); i++ {
                    db.DB.Where("id = ?", products[i].CategoryId).Find(&category)
                    db.DB.Where("id = ?",
products[i].DiscountId).Limit(intLimit).Offset(intSkip).Find(&discount).Count(&count)
                    db.DB.Find(&ratings, "product_rating")
                    db.DB.Model(&rating).Select("AVG(product_rating)").Where("product_id = ?",
products[i].Id).Scan(&averageRating)
                    count = int64(len(products))
                    productsRes = append(productsRes,
                        &models.ProductResult{
                            Id:            products[i].Id,
                            Sku:           products[i].Sku,
                            Name:          products[i].Name,
                            Stock:         products[i].Stock,
                            Price:         products[i].Price,
                            Image:         products[i].Image,
                            Category:      category,
                            Discount:      discount,
                            ProductRating: averageRating,
                        },
                    )
```

```
                }
                meta := map[string]interface{}{
                        "total":  count,
                        "Rating": averageRating,
                        "limit":  limit,
                        "skip":   skip,
                }
                return c.Status(200).JSON(fiber.Map{
                        "success": true,
                        "message": "Success",
                        "data": map[string]interface{}{
                                "products": productsRes,
                                "meta":     meta,
                        },
                })
        }
}
```

2) Comenting:
Create Comment

```go
func CreateAnotherComment(c *fiber.Ctx) error {  1 usage  rustem
    var data NewComment
    err := c.BodyParser(&data)
    if err != nil {
        log.Fatalf("Product error in post request #{err}")
    }
    var p []models.Product
    //var cashier models.Cashier
    db.DB.Find(&p)
    comment := models.Comment{
        CashierId: data.Cashier,
        ProductId: data.Product,
        //CashierId: data.CashierId,
        //ProductId: data.ProductId,
        Content:   data.Content,
        CreatedAt: time.Time{},
        UpdatedAt: time.Time{},
    }
    db.DB.Create(&comment)

    //db.DB.Table("comments").Where("id = ?", comment.Id).Update("sku", "SK00"+strconv.Itoa(comme

    Response := map[string]interface{}{
        "success": true,
        "message": "Success",
        "data":    comment,
    }
    return (c.JSON(Response))

}
```

Get comments

```go
func CommentsList(c *fiber.Ctx) error {

        limit, _ := strconv.Atoi(c.Query("limit"))

        skip, _ := strconv.Atoi(c.Query("skip"))

        var count int64

        var comment []models.Comment

        db.DB.Select("*").Limit(limit).Offset(skip).Find(&comment).Count(&count)

        type CommentList struct {

                CommentId int           `json:"commentId"`

                CashierID int           `json:"cashiersId"`

                ProductID int           `json:"productId"`
```

```
		Content   string         `json:"content"`
		CreatedAt time.Time      `json:"createdAt"`
		Cashiers  models.Cashier `json:"cashier"`
		Product   models.Product `json:"product"`
	}
	CommentResponse := make([]*CommentList, 0)
	for _, v := range comment {
		cashier := models.Cashier{}
		db.DB.Where("id = ?", v.CashierId).Find(&cashier)
		product := models.Product{}
		db.DB.Where("id = ?", v.ProductId).Find(&product)
		CommentResponse = append(CommentResponse, &CommentList{
			CommentId: v.Id,
			CashierID: v.CashierId,
			ProductID: v.ProductId,
			Content:   v.Content,
			CreatedAt: v.CreatedAt,
			Cashiers:  cashier,
			Product:   product,
		})
	}
	return c.Status(404).JSON(fiber.Map{
		"success": true,
		"message": "Sucess",
		"data":    CommentResponse,
		"meta": map[string]interface{}{
			"total": count,
			"limit": limit,
			"skip":  skip,
		},
	})
}
```

3) Give rating:

Create rating

```go
func CreateRating(c *fiber.Ctx) error {  1 usage  rustem
    var data RatingStruct
    err := c.BodyParser(&data)
    if err != nil {
        log.Fatalf("Product error in post request #{err}")
    }

    rating := models.Rating{
        CashierId:     data.CashierId,
        ProductId:     data.ProductId,
        ProductRating: data.ProdRating,
        CreatedAt:     time.Time{},
        UpdatedAt:     time.Time{},
    }
    db.DB.Create(&rating)

    Response := map[string]interface{}{
        "success": true,
        "message": "Success",
        "data":    rating,
    }
    return (c.JSON(Response))

}
```

RatingList:

```go
func RatingList(c *fiber.Ctx) error {  1 usage  rustem
    limit, _ := strconv.Atoi(c.Query( key: "limit"))
    skip, _ := strconv.Atoi(c.Query( key: "skip"))
    var count int64
    var rating []models.Rating

    db.DB.Select( query: "*").Limit(limit).Offset(skip).Find(&rating).Count(&count)

    type RatingList struct {
        RatingId   int            `json:"ratingId"`
        CashierID  int            `json:"cashiersId"`
        ProductID  int            `json:"productId"`
        ProdRating float64        `json:"prodRating"`
        CreatedAt  time.Time      `json:"createdAt"`
        Cashiers   models.Cashier `json:"cashier"`
        Product    models.Product `json:"product"`
    }
    RatingsResponse := make([]*RatingList, 0)

    for _, v := range rating {
        cashier := models.Cashier{}
        db.DB.Where( query: "id = ?", v.CashierId).Find(&cashier)
        product := models.Product{}
        db.DB.Where( query: "id = ?", v.ProductId).Find(&product)

        RatingsResponse = append(RatingsResponse, &RatingList{
            RatingId:   v.Id,
            CashierID:  v.CashierId,
            ProductID:  v.ProductId,
            ProdRating: float64(v.ProductRating),
            CreatedAt:  v.CreatedAt,
            Cashiers:   cashier,
            Product:    product,
        })
```