ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG
BỘ MÔN MÁY TÍNH - HỆ THỐNG NHÚNG

NGÔ TRẦN GIA THỊNH
MSSV: 1620240

Đề tài:

XÂY DỰNG WEB APP QUIZ

ĐỒ ÁN TỐT NGHIỆP
NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHUYÊN NGÀNH MÁY TÍNH - HỆ THỐNG NHÚNG

NGƯỜI HƯỚNG DẪN
ThS. ĐỖ QUỐC MINH ĐĂNG

TP. Hồ Chí Minh, tháng 07 năm 2020

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

STT	Kí hiệu viết tắt	Nội dung viết tắt
1	API	Application Programming Interface
2	JSON	JavaScript Object Notation
3	TOML	Tom's Obvious, Minimal Language
4	YAML	Ain’t Markup Language
5	HCL	HashiCorp Configuration Language
6	ORM	Object Relational Mapping
7	SQL	Structured Query Language
8	CRUD	Create, Read, Update, Delete
9	JWT	JSON Web Token
10	HMAC	Hash-based Message Authentication Code
11	RSA	Rivest Shamir Adleman
12	HTTP	HyperText Transfer Protocol
13	URL	Uniform Resource Locator
14	UI	User Interface
15	JSX	JavaScript XML
16	ECMAScript	European Computer Manufacturers Association Script
17	SASS/ SCSS	Syntactically Awesome Style Sheets
18	CSS	Cascading Style Sheets
19	RDBMS	Relational Database Management System

MỤC LỤC HÌNH

LỜI CẢM ƠN

Để có thể hoàn thành đề tài một cách hoàn chỉnh, bên cạnh sự nỗ lực bản thân còn có sự hướng dẫn nhiệt tình của quý Thầy Cô, cũng như sự động viên ủng hộ của gia đình bạn bè trong suốt thời gian học tập và nghiên cứu đồ án tốt nghiệp.

Tôi xin chân thành cảm ơn thầy Đỗ Quốc Minh Đăng, giảng viên chuyên ngành Máy tính – Hệ thống nhúng đã hướng dẫn và chỉ bảo tận tình cho tôi hoàn thành đồ án tốt nghiệp này.

Tôi cũng xin gửi lời cảm ơn đến các thầy, cô trong khoa Điện tử viễn thông, trường Đại học Khoa Học Tự Nhiên thành phố Hồ Chí Minh, những người đã giúp đỡ, tạo điều kiện cho tôi trong suốt quá trình học tập và nghiên cứu.

Cuối cùng tôi cũng xin cảm ơn tới gia đình, bạn bè đã giúp đỡ, động viên tôi rất nhiều trong suốt quá trình học tập để tôi có thể thực hiện tốt đồ án tốt nghiệp này. Mặc dù cố gắng hoàn thành đồ án trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những thiếu sót.

Mong nhận được sự thông cảm, góp ý và tận tính chỉ bảo của quý thầy cô và các bạn.

Hồ Chí Minh, tháng 07 năm 2020

Sinh viên

Ngô Trần Gia Thịnh

TÓM TẮT

Kiến thức là sức mạnh. Có rất nhiều điều trên thế giới cần tìm hiểu và các câu đố giúp kiểm tra sự hiểu biết về các khái niệm đó. Cũng chính vì điều đó là Web Application Quiz ra đời. Ứng dụng sẽ đưa ra các câu hỏi và mong đợi câu trả lời đúng cho các câu hỏi đó. Với Web Application Quiz, con người vừa có thể học tập vừa có thể giải trí cùng nhau. Chính vì vậy mà em chọn đề tài này.

Mục đích của đề tài này: Nguyên cứu công nghệ và xây dựng một web app tạo ra các câu hỏi và câu trả lời để tổng hợp tất cả kiến thức dưới dạng đố vui có thưởng. Qua đó người sử dụng có thể biết thêm nhiều kiến thức cũng như là cũng cố những kiến thức đã biết. Ứng dụng sử dụng Gin Web Framework (được viết bằng Golang) để làm máy chủ, sử dụng ReactJS cùng với các thư viện bổ trợ để làm giao diện người dùng.

Nội dung trong cuốn báo cáo gồm:

- Phần lý thuyết: Trình bày về các công nghệ và các thư viện được sử dụng để thực hiện đề tài này.
- Phần thực hiện: Trình bày về các luồng chạy của ứng dụng. Các chức năng và màn hình trong ứng dụng.
- Phần kết luận và hướng phát triển: Trình bày về những gì đã đạt được khi thực hiện đề tài này. Qua đó cũng nhìn nhân được những thiếu sót để làm tiền đề phát triển sau này.
- Phần tài liệu tham khảo: Trình bày về các tài liệu tham khảo khi thực hiện đề tài.

CHƯƠNG 1. LÝ THUYẾT

Các công nghệ và thư viện được sử dụng cho backend:

- Gin Web Framework: được sử dụng để xây dựng bộ sườn cho hệ thống. Việc sử dụng công nghệ này sẽ giúp tiết kiêm thời gian.
- Spf13/Viper: sử dụng cho mục đích lưu trữ các cấu hình hệ thống dưới định dạng yml hoặc json.
- Gorm: sử dụng cho việc giảm thiểu viết các câu query, giúp cho việc giao tiếp giữa máy chủ và cơ sỡ dữ liệu dễ dàng hơn.
- JWT và Redis: JWT được dùng để định danh người dùng bằng những đoạn token. Redis dùng để lưu trữ các token của JWT trong bộ nhớ và trong một khoảng thời gian nhất định.

Các công nghệ và thư viện được sử dụng cho frontend:

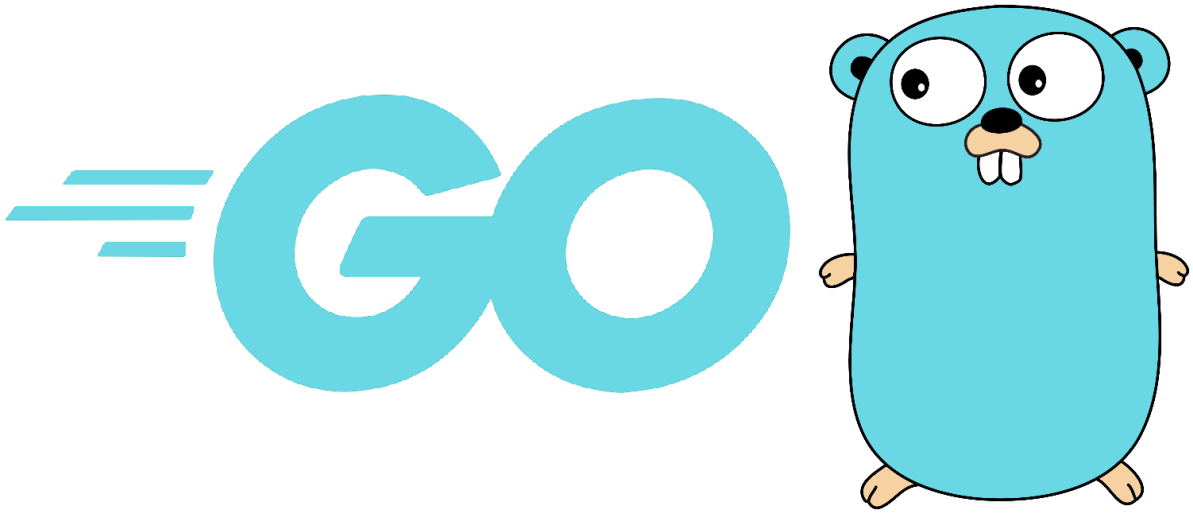
- ReactJS: Được sử dụng để xây dựng giao diện người dùng. Việc sử dụng thư viên này sẽ giúp trang web không bị tải trang, qua đó sẽ làm tăng trải nghiệm của người dùng
- React-router: Là bộ thư viên bộ trợ cho ReactJS. Công dụng của nó là giúp người dùng chuyển trang.
- React-spring: Là bộ thư viện bổ trợ cho ReactJS. Cộng dụng của nó là tạo ra các hiệu ứng giúp người dùng trải nghiệm mượt mà hơn.
- Cookie-universal: Đùng để lưu trữ các cookie từ phía người dùng.

1.1 Golang

Go là một ngôn ngữ lập trình mới do Google thiết kế và phát triển. Nó được kỳ vọng sẽ giúp ngành công nghiệp phần mềm khai thác nền tảng đa lõi của bộ vi xử lý và hoạt động đa nhiệm tốt hơn. Go là một ngôn ngữ lập trình được thiết kế dựa trên tư duy lập trình hệ thống. Go được phát triển bởi Robert Griesemer, Rob Pike và Ken Thompson tại Google vào năm 2007. Điểm mạnh của Go là bộ thu gom rác và hỗ trợ lập trình đồng thời (tương tự như đa luồng – multithreading). Go là một ngôn ngữ biên dịch như C/C++, Java, Pascal… Go được giới thiệu vào năm 2009 và được sử dụng hầu hết trong các sản phẩm của Google. Go là một dự án mã nguồn mở.

Go có một số đặt tính như:

- Hỗ trợ khai báo kiểu dữ liệu động.
- Tốc độ biên dịch nhanh.
- Hỗ trợ các tác vụ đồng thời.
- Ngôn ngữ đơn giản, ngắn gọn.
- Không hỗ trợ kế thừa.
- Không hỗ trợ thao tác trên con trỏ.
- Không hỗ trợ kiểu Generic.



Hình 1.1 Ngôn ngữ Go

1.2 Gin Web Framework

Gin là một web framework được viết bằng Golang, hoạt đông tương tự Martini API. Martini cũng là web framework được biết bằng Golang nhưng đã ngừng hỗ trợ từ lâu. Gin được kế thừa từ Martini và cho ra hiệu suất vượt trội nhanh gấp 40 lần. Nó sử dụng một phiên bản tùy biến của gói httprouter vì tốc độ xử lí cực kì nhanh, điều này làm cho nó vô cùng hoàn hảo để phát triển API hiệu xuất cao. Song song đó, nó cung cấp các trình xử lý cho nhiều trường hợp sử dụng phổ biến: middleware, file uploading, logging, binding front-end HTML component với cấu trúc dữ liệu back-end, …

1.2.1 Api Example

Hình dưới chỉ ra các cách khai báo phương thức http cơ bản:

```go
func main() {
        // Creates a gin router with default middleware:
        // logger and recovery (crash-free) middleware
        router := gin.Default()

        router.GET("/someGet", getting)
        router.POST("/somePost", posting)
        router.PUT("/somePut", putting)
        router.DELETE("/someDelete", deleting)
        router.PATCH("/somePatch", patching)
        router.HEAD("/someHead", head)
        router.OPTIONS("/someOptions", options)

        // By default it serves on :8080 unless a
        // PORT environment variable was defined.
        router.Run()
        // router.Run(":3000") for a hard coded port
}
```

Hình 1.2 Các phương thức cơ bản

1.2.2 Parameters In Path

```go
func main() {
	router := gin.Default()

	// This handler will match /user/john but will not match /user/ or /user
	router.GET("/user/:name", func(c *gin.Context) {
		name := c.Param("name")
		c.String(http.StatusOK, "Hello %s", name)
	})

	// However, this one will match /user/john/ and also /user/john/send
	// If no other routers match /user/john, it will redirect to /user/john/
	router.GET("/user/:name/*action", func(c *gin.Context) {
		name := c.Param("name")
		action := c.Param("action")
		message := name + " is " + action
		c.String(http.StatusOK, message)
	})

	// For each matched request Context will hold the route definition
	router.POST("/user/:name/*action", func(c *gin.Context) {
		c.FullPath() == "/user/:name/*action" // true
	})

	router.Run(":8080")
}
```

Hình 1.3 Lấy các thông số param

1.2.3 Querystring Parameters

```go
func main() {
	router := gin.Default()

	// Query string parameters are parsed using the existing underlying request object.
	// The request responds to a url matching:  /welcome?firstname=Jane&lastname=Doe
	router.GET("/welcome", func(c *gin.Context) {
		firstname := c.DefaultQuery("firstname", "Guest")
		lastname := c.Query("lastname") // shortcut for c.Request.URL.Query().Get("lastname")

		c.String(http.StatusOK, "Hello %s %s", firstname, lastname)
	})
	router.Run(":8080")
}
```

Hình 1.4 Lấy thông số string query

1.2.4 Upload Files

- Single file

```
func main() {
        router := gin.Default()
        // Set a lower memory limit for multipart forms (default is 32 MiB)
        router.MaxMultipartMemory = 8 << 20  // 8 MiB
        router.POST("/upload", func(c *gin.Context) {
                // single file
                file, _ := c.FormFile("file")
                log.Println(file.Filename)

                // Upload the file to specific dst.
                c.SaveUploadedFile(file, dst)

                c.String(http.StatusOK, fmt.Sprintf("'%s' uploaded!", file.Filename))
        })
        router.Run(":8080")
}
```

Hình 1.5. Cách tải một tập tin lên máy chủ

- Multiple files

```
func main() {
        router := gin.Default()
        // Set a lower memory limit for multipart forms (default is 32 MiB)
        router.MaxMultipartMemory = 8 << 20  // 8 MiB
        router.POST("/upload", func(c *gin.Context) {
                // Multipart form
                form, _ := c.MultipartForm()
                files := form.File["upload[]"]

                for _, file := range files {
                        log.Println(file.Filename)

                        // Upload the file to specific dst.
                        c.SaveUploadedFile(file, dst)
                }
                c.String(http.StatusOK, fmt.Sprintf("%d files uploaded!", len(files)))
        })
        router.Run(":8080")
}
```

Hình 1.6. Cách tải nhiều tập tin lên máy chủ

1.2.5 Grouping Routes

```
func main() {
        router := gin.Default()

        // Simple group: v1
        v1 := router.Group("/v1")
        {
                v1.POST("/login", loginEndpoint)
                v1.POST("/submit", submitEndpoint)
                v1.POST("/read", readEndpoint)
        }

        // Simple group: v2
        v2 := router.Group("/v2")
        {
                v2.POST("/login", loginEndpoint)
                v2.POST("/submit", submitEndpoint)
                v2.POST("/read", readEndpoint)
        }

        router.Run(":8080")
}
```

Hình 1.7. Phân nhóm các đường dẫn

1.2.6 Using Middleware

Không có middleware:

```
r := gin.New()
```

Hình 1.8. Tạo máy chủ không sử dụng phương thức trung gian

Với default middleware:

```
// Default With the Logger and Recovery middleware already attached
r := gin.Default()
```

Hình 1.9. Tạo máy chủ sử dụng phương thức trung gian

Ghi log ra file:

```go
func main() {
    // Disable Console Color, you don't need console color when writing the logs to file.
    gin.DisableConsoleColor()

    // Logging to a file.
    f, _ := os.Create("gin.log")
    gin.DefaultWriter = io.MultiWriter(f)

    // Use the following code if you need to write the logs to file and console at the same time.
    // gin.DefaultWriter = io.MultiWriter(f, os.Stdout)

    router := gin.Default()
    router.GET("/ping", func(c *gin.Context) {
        c.String(200, "pong")
    })

    router.Run(":8080")
}
```

Hình 1.10 Cách ghi log ra tập tin

Tùy chỉnh một định dạng log.

```go
func main() {
        router := gin.New()

        // LoggerWithFormatter middleware will write the logs to gin.DefaultWriter
        // By default gin.DefaultWriter = os.Stdout
        router.Use(gin.LoggerWithFormatter(func(param gin.LogFormatterParams) string {

                // your custom format
                return fmt.Sprintf("%s - [%s] \"%s %s %s %d %s \"%s\" %s\"\n",
                                param.ClientIP,
                                param.TimeStamp.Format(time.RFC1123),
                                param.Method,
                                param.Path,
                                param.Request.Proto,
                                param.StatusCode,
                                param.Latency,
                                param.Request.UserAgent(),
                                param.ErrorMessage,
                )
        }))
        router.Use(gin.Recovery())

        router.GET("/ping", func(c *gin.Context) {
                c.String(200, "pong")
        })

        router.Run(":8080")
}
```

Hình 1.11 Cách ghi log với thông tin tùy chọn

1.2.7 Model Binding And Validation

```
// Example for binding JSON ({"user": "manu", "password": "123"})
router.POST("/loginJSON", func(c *gin.Context) {
        var json Login
        if err := c.ShouldBindJSON(&json); err != nil {
                c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
                return
        }

        if json.User != "manu" || json.Password != "123" {
                c.JSON(http.StatusUnauthorized, gin.H{"status": "unauthorized"})
                return
        }

        c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
})
```

Hình 1.12 Ràng buộc vào model bằng định dạng json

```
// Example for binding a HTML form (user=manu&password=123)
router.POST("/loginForm", func(c *gin.Context) {
        var form Login
        // This will infer what binder to use depending on the content-type header.
        if err := c.ShouldBind(&form); err != nil {
                c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
                return
        }

        if form.User != "manu" || form.Password != "123" {
                c.JSON(http.StatusUnauthorized, gin.H{"status": "unauthorized"})
                return
        }

        c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
})
```

Hình 1.13. Ràng buộc vào model bằng parameter

1.3 Spf13/Viper

Viper là một thư viện hoàn chỉnh cho việc cấu hình trong các dự án Golang. Nó được thiết kế để có thể xử lý được hết tất cả các định dạng. Viper hỗ trợ:

- Tùy chỉnh mặc định
- Đọc từ các format JSON, TOML, YAML, HCL, envfile và java config.

- Xem trực tiếp và đọc các tập tin cấu hình.
- Đọc từ các biến môi trường.
- Đọc từ các hệ thống cấu hình từ xa.
- Đoc từ cmd.
- Đọc từ bộ đêm (buffer).

Hình 1.14. Viper

1.4 Gorm

Go là một ngôn ngữ dựa trên sự đơn giản và tốc độ. Tuy nhiên sự phức tạp của một ứng dụng có thể làm tăng đáng kể sự phải tương tác với cơ sở dữ liệu. Gorm là một thư viện cung cấp khả năng ánh xạ để có thể loại bỏ phần lớn sự phức tạp này. Nó có một số đặt điểm sau:

- Gần đầy đủ hết tất cả tính năng ORM.
- Associations (Has one, has many, belongs to, many to many, đa hình)
- Hooks
- Preloading
- Transactions
- Tạo khóa ngoại.
- SQL builder.
- Auto Migrations.
- Logger.
- Có thể mở rộng, viết plugin đựa trên Gorm callback function.
- Thân thiện với lập trình viên.

1.4.1 Declaring Models

Models trong GORM là Golang struct.

```go
type User struct {
  gorm.Model
  Name         string
  Age          sql.NullInt64
  Birthday     *time.Time
  Email        string  `gorm:"type:varchar(100);unique_index"`
  Role         string  `gorm:"size:255"` // set field size to 255
  MemberNumber *string `gorm:"unique;not null"` // set member number to unique and not null
  Num          int     `gorm:"AUTO_INCREMENT"` // set num to auto incrementable
  Address      string  `gorm:"index:addr"` // create index with name `addr` for address
  IgnoreMe     int     `gorm:"-"` // ignore this field
}
```

Hình 1.15. Kiểu struct trong Go

Các thẻ:

Column	Xác định tên cột
Type	Kiểu dữ liệu
Size	Giới gian, mặc định 255
PRIMARY_KEY	Khóa chính
UNIQUE	Giá trị cột là độc nhất
DEFAULT	Giá trị mặc định cho cột
PRECISION	Độ chính xác
NOT NULL	Ràng buộc giá trị cột là NOT NULL
AUTO_INCREMENT	Cột tự tăng
INDEX	Khởi tạo giá trị index trong cột
UNIQUE_INDEX	Tạo giá trị index độc nhất
EMBEDDED	Kế thừa struct trong Go
EMBEDDED_PREFIX	Kết thừa tiếp đầu ngữ của struct
-	Bỏ qua giá trị này

Các thẻ liên kết:

MANY2MANY	Tên bản được liên kết
FOREIGNKEY	Khóa ngoại
ASSOCIATION_FOREIGNKEY	Khóa ngoại liên kết
POLYMORPHIC	Kiểu đa hình
POLYMORPHIC_VALUE	Giá trị đa hình
JOINTABLE_FOREIGNKEY	Xác định khóa ngoại có thể liên kết
ASSOCIATION_JOINTABLE_FOREIGNKEY	Xác định liên kết của khóa ngoại có thể liên kết
SAVE_ASSOCIATIONS	Tự động lưu liên kết hoặc không
ASSOCIATION_AUTOUPDATE	Tự đông cập nhật liên kết hoặc không
ASSOCIATION_AUTOCREATE	Tự động tạo liên kết hoặc không

ASSOCIATION_SAVE_REFERENCE	Tự động lưu liên kết tham khảo hoặc không
PRELOAD	Tự động tải lại liên kết hoặc không

Gorm.model là một struct cơ bản bao gồm các trường: ID, CreateAt, UpdateAt, DeleteAt. Nó có thể được nhúng vào model hoặc không:

```
// gorm.Model definition
type Model struct {
  ID        uint `gorm:"primary_key"`
  CreatedAt time.Time
  UpdatedAt time.Time
  DeletedAt *time.Time
}

// Inject fields `ID`, `CreatedAt`, `UpdatedAt`, `DeletedAt` into model `User`
type User struct {
  gorm.Model
  Name string
}

// Declaring model w/o gorm.Model
type User struct {
  ID   int
  Name string
}
```

Hình 1.16. Khai báo model

Trong đó GORM mặc định sử dụng ID là primary key. Các table khi tạo ra sẽ ở dạng số nhiều và được ngăn cách nhau bằng dấu gạch. VD: model là ModelUser thì table được tạo ra sẽ là model_users.

Các dấu thời gian được thể hiện ở ba trường: CreateAt, UpdatedAt, DeleteAt. CreateAt sẽ được khởi tạo khi bảng ghi được tạo lần đầu tiên. UpdatedAt sẽ được tạo khi bảng ghi được cập nhật và DeleteAt được khởi tạo khi phương thức Delete được gọi tới.

1.4.2 CRUD Interface

Tạo bảng ghi:

```
user := User{Name: "Jinzhu", Age: 18, Birthday: time.Now()}

db.NewRecord(user) // => returns `true` as primary key is blank

db.Create(&user)

db.NewRecord(user) // => return `false` after `user` created
```

Hình 1.17. Tạo bảng ghi

Query:

```
// Get first record, order by primary key
db.First(&user)
//// SELECT * FROM users ORDER BY id LIMIT 1;

// Get one record, no specified order
db.Take(&user)
//// SELECT * FROM users LIMIT 1;

// Get last record, order by primary key
db.Last(&user)
//// SELECT * FROM users ORDER BY id DESC LIMIT 1;

// Get all records
db.Find(&users)
//// SELECT * FROM users;

// Get record with primary key (only works for integer primary key)
db.First(&user, 10)
//// SELECT * FROM users WHERE id = 10;
```

Hình 1.18. Query trong Gorm

```
// Get first matched record
db.Where("name = ?", "jinzhu").First(&user)
//// SELECT * FROM users WHERE name = 'jinzhu' ORDER BY id LIMIT 1;

// Get all matched records
db.Where("name = ?", "jinzhu").Find(&users)
//// SELECT * FROM users WHERE name = 'jinzhu';

// <>
db.Where("name <> ?", "jinzhu").Find(&users)
//// SELECT * FROM users WHERE name <> 'jinzhu';

// IN
db.Where("name IN (?)", []string{"jinzhu", "jinzhu 2"}).Find(&users)
//// SELECT * FROM users WHERE name in ('jinzhu','jinzhu 2');

// LIKE
db.Where("name LIKE ?", "%jin%").Find(&users)
//// SELECT * FROM users WHERE name LIKE '%jin%';

// AND
db.Where("name = ? AND age >= ?", "jinzhu", "22").Find(&users)
//// SELECT * FROM users WHERE name = 'jinzhu' AND age >= 22;

// Time
db.Where("updated_at > ?", lastWeek).Find(&users)
//// SELECT * FROM users WHERE updated_at > '2000-01-01 00:00:00';

// BETWEEN
db.Where("created_at BETWEEN ? AND ?", lastWeek, today).Find(&users)
//// SELECT * FROM users WHERE created_at BETWEEN '2000-01-01 00:00:00' AND '2000-01-08 00:0
```

Hình 1.19. Query trong Gorm

Khi viết query với struct, GORM chỉ query những trường khác không. Điều đó nghĩa là các trường 0, ‘’, false sẽ không có hiệu lực trong câu query.

Select:

```
db.Select("name, age").Find(&users)
//// SELECT name, age FROM users;

db.Select([]string{"name", "age"}).Find(&users)
//// SELECT name, age FROM users;

db.Table("users").Select("COALESCE(age,?)", 42).Rows()
//// SELECT COALESCE(age,'42') FROM users;
```

Hình 1.20. Query trong Gorm

Order:

```
db.Order("age desc, name").Find(&users)
//// SELECT * FROM users ORDER BY age desc, name;

// Multiple orders
db.Order("age desc").Order("name").Find(&users)
//// SELECT * FROM users ORDER BY age desc, name;

// ReOrder
db.Order("age desc").Find(&users1).Order("age", true).Find(&users2)
//// SELECT * FROM users ORDER BY age desc; (users1)
//// SELECT * FROM users ORDER BY age; (users2)
```

Hình 1.21. Query trong Gorm

Limit:

```
db.Limit(3).Find(&users)
//// SELECT * FROM users LIMIT 3;

// Cancel limit condition with -1
db.Limit(10).Find(&users1).Limit(-1).Find(&users2)
//// SELECT * FROM users LIMIT 10; (users1)
//// SELECT * FROM users; (users2)
```

Hình 1.22. Query trong Gorm

Offset:

```
db.Offset(3).Find(&users)
//// SELECT * FROM users OFFSET 3;

// Cancel offset condition with -1
db.Offset(10).Find(&users1).Offset(-1).Find(&users2)
//// SELECT * FROM users OFFSET 10; (users1)
//// SELECT * FROM users; (users2)
```

Hình 1.23. Query trong Gorm

Joins:

```
db.Table("users").Select("users.name, emails.email").Joins("left join emails on emails.user_id = users.id").Scan(&results)
```

Hình 1.24. Query trong Gorm

Xóa bảng ghi:

Khi xóa bảng ghi, phải chắc chắn rằng primary key có giá trị và GORM sử dụng primary key để xóa bảng ghi. Nếu primary key bỏ trống, GORM sẽ xóa toàn bộ bảng ghi trong model.

```
// Delete an existing record
db.Delete(&email)
//// DELETE from emails where id=10;

// Add extra SQL option for deleting SQL
db.Set("gorm:delete_option", "OPTION (OPTIMIZE FOR UNKNOWN)").Delete(&email)
//// DELETE from emails where id=10 OPTION (OPTIMIZE FOR UNKNOWN);
```

Hình 1.25. Delete trong Gorm

Nếu model có trường DeleteAt, thì nó sẽ được xóa mềm. Khi gọi phương thức Delete, bản ghi sẽ không được xóa khỏi cơ sở dữ liệu một cách vĩnh viễn, thay vào đó DeleteAt sẽ được khởi tạo giá trị là ngày xóa.

```
db.Delete(&user)
//// UPDATE users SET deleted_at="2013-10-29 10:23" WHERE id = 111;

// Batch Delete
db.Where("age = ?", 20).Delete(&User{})
//// UPDATE users SET deleted_at="2013-10-29 10:23" WHERE age = 20;

// Soft deleted records will be ignored when query them
db.Where("age = 20").Find(&user)
//// SELECT * FROM users WHERE age = 20 AND deleted_at IS NULL;

// Find soft deleted records with Unscoped
db.Unscoped().Where("age = 20").Find(&users)
//// SELECT * FROM users WHERE age = 20;
```

Hình 1.26. Delete trong Gorm

Ngược lại, muốn xóa vĩnh viên khỏi cơ sở dữ liệu, ta sẽ dùng:

```
// Delete record permanently with Unscoped
db.Unscoped().Delete(&order)
//// DELETE FROM orders WHERE id=10;
```

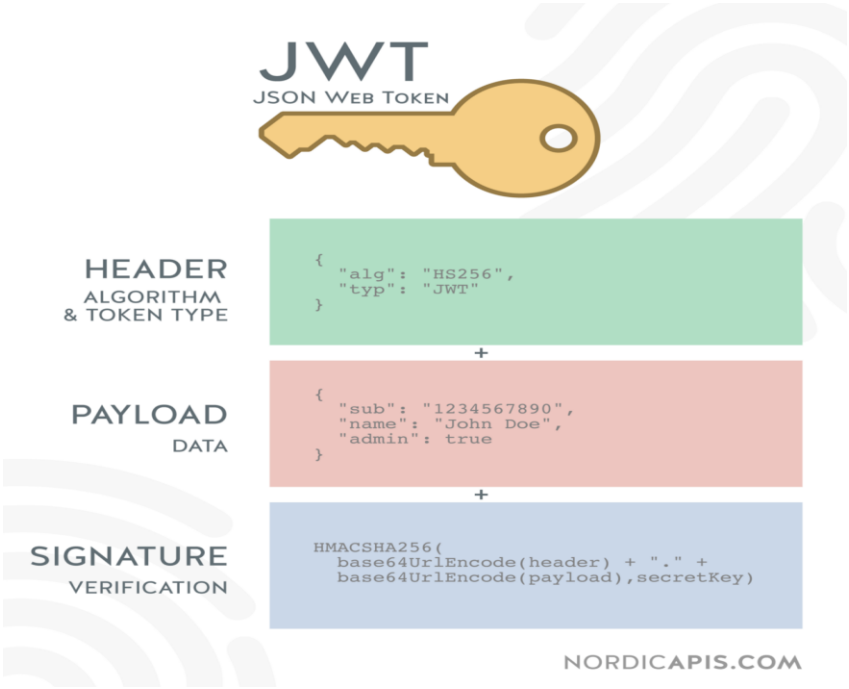
Hình 1.27. Delete trong Gorm

1.5 JWT Và Redis

JSON Web Token (JWT) là một tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng một đối tượng JSON. Thông tin này có thể được xác thực và đánh dấu tin cậy nhờ vào "chữ ký" của nó. Phần chữ ký của JWT sẽ được mã hóa lại bằng HMAC hoặc RSA.

Những đặc điểm nổi bật của JWT:

- Kích thước nhỏ: JWT có thể được truyền thông qua URL, hoặc qua giao thức POST, hay nhét vào bên trong phần HTTP Header. Kích thước nhỏ hơn ứng với công việc truyền tải sẽ nhanh hơn.
- Khép kín: Phần Payload (hiểu nôm na là khối hàng) chứa toàn bộ những thông tin mà chúng ta cần tới, ví dụ như thông tin của người dùng (thay vì phải truy vấn cơ sở dữ liệu nhiều lần).



Hình 1.28. JWT

Redis (REmote DIctionary Server) là một mã nguồn mở được dùng để lưu trữ dữ liệu có cấu trúc, có thể sử dụng như một cơ sở dữ liệu, bộ nhớ cache hay một message broker. Nó là hệ thống lưu trữ dữ liệu với dạng KEY-VALUE rất mạnh mẽ và phổ biến hiện nay. Redis nổi bật bởi việc hỗ trợ nhiều cấu trúc dữ liệu cơ bản như: hash, list, set, sorted set, string… Tất cả dữ liệu được ghi và lưu trên RAM, do đó tốc độ đọc ghi dữ liệu rất là nhanh.

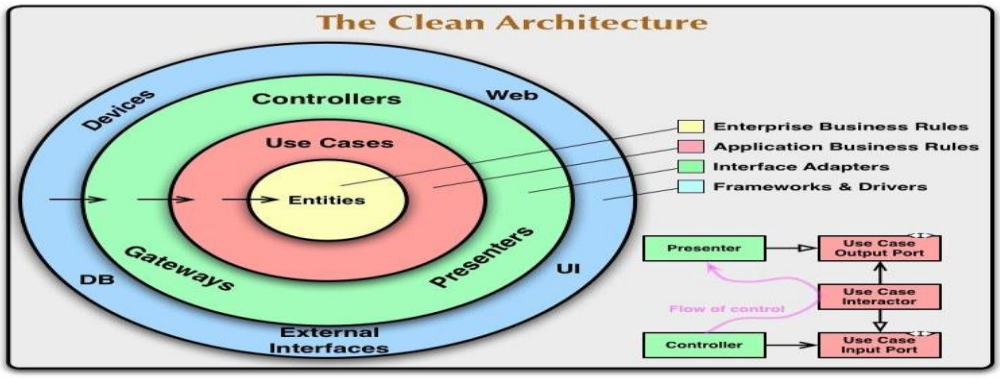
Redis ngoài tính năng lưu trữ KEY-VALUE trên RAM thì Redis còn hỗ trợ tính năng xắp xếp, query, backup dữ liệu trên đĩa cứng cho phép bạn có thể phục hồi dữ liệu khi hệ thống gặp sự cố…và có thể nhân bản (Chạy nhiều Server Redis cùng lúc).

Hình 1.29. Redis

1.6 Clean Architecture (Uncle Bob)

Clean Architecture được xây dựng dựa trên tư tưởng "độc lập" kết hợp với các nguyên lý thiết kế hướng đối tượng(đại diện tiêu biểu là Dependency Inversion). Độc lập ở đây nghĩa là việc project không bị phụ thuộc vào framework và các công cụ sử dụng trong quá trình kiểm thử.

Kiến trúc của Clean Architecture chia thành bốn layer với một quy tắc phụ thuộc. Các layer bên trong không nên biết bất kỳ điều gì về các layer bên ngoài. Điều này có nghĩa là nó có quan hệ phụ thuộc nên "hướng" vào bên trong. Nhìn vào hình vẽ minh họa sau đây:



Hình 1.30. Clean Architecture

Entities: là khái niệm dùng để mô tả các Business Logic. Đây là layer quan trọng nhất, là nơi bạn thực hiện giải quyết các vấn đề - mục đích khi xây dựng app. Layer này không chứa bất kỳ một framework nào, nó có thể chạy nó mà không cần emulator. Nó giúp dễ dàng test, maintain và develop phần business logic.

Use case: chứa các rule liên quan trực tiếp tới ứng dụng cục bộ (application-specific business rules).

Interface Adapter: tập hợp các adapter phục vụ quá trình tương tác với các công nghệ.

Framework and Drivers: chứa các công cụ về cơ sở dữ liệu và các framework, thông thường bạn sẽ không phải lập trình nhiều ở tầng này. Tuy nhiên cần chắc chắn về mức ưu tiên sử dụng các công cụ này trong project.

Clean Architecture mang lại những lợi ích sau:

- Mạch lạc - dễ xem (bản gốc ghi screaming với dụng ý là chỉ cần nhìn cấu trúc package cũng có thể hiểu được mục đích và cơ chế hoạt động của ứng dụng).
- Linh hoạt - thể hiện ở khả năng độc lập, không phụ thuộc vào framework, database, application server.
- Dễ kiểm thử - testable.

Bên cạnh những lợi ích trên thì Clean Architecture còn những hạn chế sau:

- Không thể sử dụng framework theo cách mỳ ăn liền- do luật dependency inversion.
- Khó áp dụng.
- Cồng kềnh - thể hiện ở việc có quá nhiều class so với các project cùng mục tiêu (tuy nhiên các class được thêm vào đều có chủ ý và đáp ứng đúng quy định khi triển khai kiến trúc).

1.7 ReactJs

Được phát triển Facebook, React là một thư viện javascript phổ biến nhất hiện nay dùng để xây dựng UI (User Interface) có tính tương tác cao, có trạng thái và có thể sử

dụng lại được. Điển hình như Facebook và Instagram là nơi làm ra và maintain React.js thì Yahoo hay Airbnb là những ví dụ nổi bật có sử dụng thư viện này.

Điểm thu hút của React chính là việc nó không chỉ hoạt động trên phía client mà còn được render trên server và có thể kết nối với nhau.

Virtual DOM: Virtual DOM là một object Javascript, mội object chứa đầy đủ các thông tin cần thiết để tạo ra một DOM, nó sẽ giúp tối ưu hóa việc re-render DOM tree thật bằng cách tính toán sự thay đổi giữa object và tree thật khi dữ liệu thay đổi. Việc chỉ node gốc có trạng thái và sẽ tái cấu trúc lại toàn bộ khi nó thay đổi dẫn đến việc DOM tree sẽ thay đổi một phần làm ảnh hưởng đến tốc độ xử lý. Sử dụng virtual DOM giúp cho ReactJs cải thiện vấn đề này và giúp cải thiện hiệu năng cho ứng dụng.

Giới thiệu về JSX: JSX là một dạng ngôn ngữ với ưu điểm nhanh hơn cho phép viết các mã HTML trong Javascript. Với các ưu điểm:

- Nhanh hơn: JSX giúp tối ưu hóa thời gian khi biên dịch sang mã Javascript, các đoạn mã này được thực hiện nhanh hơn so với viết trực tiếp bằng Javascript.
- An toàn hơn: khác biệt với Javascript, JSX được biên dịch trước khi chạy, giống như Java, C++ (statically-typed) giúp phát hiện lỗi ngay trong quá trình biên dịch, bên cạnh đó nó còn có tính năng gỡ lỗi khi biên dịch rất tốt.
- Dễ dàng hơn: JSX kế thừa dựa trên Javascript giúp cho các lập trình viên Javascript dễ dàng sử dụng.

Components: Khác với các framework khác, React không sử dụng template mà được xây dựng xung quanh các component. Trong React, để xây dựng trang web chúng ta sử dụng những component, có thể tái sử dụng một component ở nhiều nơi với các trạng thái hoặc thuộc tính khác nhau và một component có thể chứa thành phần khác. Mỗi component có một trạng thái riêng, có thể thay đổi và React sẽ thực hiện cập nhật component dựa trên những thay đổi của trạng thái.

Props: Giúp các component tương tác với nhau. Khi component nhận input gọi là props và trả thuộc tính mô tả những gì component con sẽ render. Đặc tính của Prop là bất biến.

State: Thể hiện trạng thái của ứng dụng. Khi state thay đổi thì component đồng thời render lại để cập nhật UI.

Hook: Hook là một khái niệm mới đc React team công bố gần đây ở React Conf 2018. Nó cho phép một functional component có thể sử dụng state, sử dụng các lifecycle method, context và nhiều thứ khác.

1.8 Typescript

TypeScript là một dự án mã nguồn mở được phát triển bởi Microsoft, nó có thể được coi là một phiên bản nâng cao của Javascript bởi việc bổ sung tùy chọn kiểu tĩnh và lớp hướng đối tượng mà điều này không có ở Javascript. TypeScript có thể sử dụng để phát triển các ứng dụng chạy ở client-side (Angular2) và server-side (NodeJS).

TypeScript sử dụng tất cả các tính năng của của ECMAScript 2015 (ES6) như classes, modules. Không dừng lại ở đó nếu như ECMAScript 2017 ra đời thì tin chắc rằng TypeScript cũng sẽ nâng cấp phiên bản của mình lên để sử dụng mọi kỹ thuật mới nhất từ ECMAScript. Thực ra TypeScript không phải ra đời đầu tiên mà trước đây cũng có một số thư viện như CoffeScript và Dart được phát triển bởi Google, tuy nhiên điểm yếu là hai thư viện này sư dụng cú pháp mới hoàn toàn, điều này khác hoàn toàn với TypeScript, vì vậy tuy ra đời sau nhưng TypeScript vẫn đang nhận được sự đón nhận từ các lập trình viên.

Ưu điểm của typescript:

- Dễ phát triển dự án lớn: Với việc sử dụng các kỹ thuật mới nhất và lập trình hướng đối tượng nên TypeScript giúp chúng ta phát triển các dự án lớn một cách dễ dàng.
- Nhiều Framework lựa chọn: Hiện nay các Javascript Framework đã dần khuyến khích nên sử dụng TypeScript để phát triển, ví dụ như AngularJS 2.0 và Ionic 2.0.
- Hỗ trợ các tính năng của Javascript phiên bản mới nhất: TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của Javascript, ví dụ như version hiện tại là ECMAScript 2015 (ES6).
- Là mã nguồn mở: TypeScript là một mã nguồn mở nên bạn hoàn toàn có thể sử dụng mà không mất phí, bên cạnh đó còn được cộng đồng hỗ trợ.
- TypeScript là Javascript: Bản chất của TypeScript là biên dịch tạo ra các đoạn mã javascript nên ban có thể chạy bất kì ở đâu miễn ở đó có hỗ trợ biên dịch Javascript. Ngoài ra bạn có thể sử dụng trộn lẫn cú pháp của Javascript vào bên trong TypeScript, điều này giúp các lập trình viên tiếp cận TypeScript dễ dàng hơn.

1.9 React-router

React Router là một thư viện điều hướng tiêu chuẩn trong React. Nó giúp cho UI được đồng bộ với URL. Nó có API đơn giản nhưng mạnh mẽ, có thể giúp giải quyết được rất nhiều vấn đề.

Các thành phần trong React-Router:

- BrowserRouter và HashRouter
- Route
- Link
- NavLink
- Redirect

1.10 React-spring

React-spring tập trung khai thác các animation vật lý, không cầu kì mà đơn giản và mang tính tổng quan (declarative), một trong những thư viện gọn và dễ tiếp cận nhất để tạo animation cho project.

1.11 Cookie-universal

Cookie-universal là thư viện dùng để thêm, sửa xóa cookie trong trình duyệt.

1.12 Axios

Axios là một HTTP client được viết dựa trên Promises được dùng để hỗ trợ cho việc xây dựng các ứng dụng API từ đơn giản đến phức tạp và có thể được sử dụng cả ở trình duyệt hay Node.js.

Việc tạo ra một HTTP request dùng để fetch hay lưu dữ liệu là một trong những nhiệm vụ thường thấy mà một ứng dụng Javascript phía client cần phải làm khi muốn giao tiếp với phía server. Các thư viện bên thứ 3, đặc biệt là jQuery từ xưa đến nay vẫn là một trong những cách phổ biến để giúp cho các browser API tương tác tốt hơn, rõ ràng mạch lạc hơn và xóa đi những điểm khác biệt giữa các browser với nhau.

1.13 Sass/Scss

SASS/SCSS là một chương trình tiền xử lý CSS (*CSS preprocessor)*. Nó giúp ta viết CSS theo cách của một ngôn ngữ lập trình, có cấu trúc rõ ràng, rành mạch, dễ phát triển và bảo trì code hơn. Ngoài ra nó có rất nhiều các thư viện hỗ trợ kèm theo giúp ta viết code CSS một cách dễ dàng vào đơn giản hơn. Có rất nhiều loại CSS Preprocessor trong đó bao gồm SASS, Stylus hay LESS.

SASS và SCSS về bản chất vấn đề là giống nhau, chỉ khác nhau ở cách viết

Sass là chữ viết tắt của Syntactically Awesome Style Sheets, chương trình tiền xử lý bằng ngôn ngữ kịch bản (Preprocessor Scripting Language), sẽ được biên dịch thành CSS. Nghĩa là, mình sẽ làm style bằng SASS, rồi SASS sẽ render việc mình làm thành file CSS.

SASS bản thân có hai kiểu viết khác nhau, một kiểu như là HAML, Pug – Sử dụng indent (cách thụt đầu dòng) để phân tách các khối code, sử dụng xuống dòng để phân biệt rules , có phần mở rộng là .sass**.**

SCSS sử dụng cú pháp giống với Ruby**.** Có phần mở rộng là .scss , SCSS ra đời sau SASS và có cú pháp viết tương tự như cách viết CSS. Cú pháp này được tạo ra nhằm thu hẹp khoảng cách giữa SASS và CSS bằng cách mang lại một thứ gì đó thân thiện với CSS.

1.14 HTTP Method

Http method, hay còn gọi là phương thức HTTP, là cách client yêu cầu server phải làm gì với request của mình. Có 9 phương thứ HTTP, nhưng phổ biến là 4 phương thức

GET, POST, PUT, DELETE tượng trưng cho 4 tác vụ CRUD (Create - Read - Update - Delete) thông thường.

Phương thức GET thường dùng để lấy dữ liệu ra từ server. GET được xem là phương thức an toàn, vì nó chỉ lấy thông tin từ server mà không làm thay đổi dữ liệu.

Phương thức POST thường được dùng để đưa dữ liệu cho server. POST thường xuất hiện trong các form html hoặc trong các rest api để khởi tạo dữ liệu. Thông thường ta cung cấp dữ liệu cho server thông qua body của request.

Phương thức PUT thường được dùng để cập nhật dữ liệu trong server.

Phương thức Delete dùng để xóa thông tin ra khỏi server.

1.15 Mysql

MySQL là một hệ thống quản trị cơ sở dữ liệu mã nguồn mở (Relational Database Management System, viết tắt là RDBMS) hoạt động theo mô hình client-server. RDBMS là một phần mềm hay dịch vụ dùng để tạo và quản lý các cơ sở dữ liệu (Database) theo hình thức quản lý các mối liên hệ giữa chúng.

MySQL là một trong số các phần mềm RDBMS. RDBMS và MySQL thường được cho là một vì độ phổ biến quá lớn của MySQL. Các ứng dụng web lớn nhất như Facebook, Twitter, YouTube, Google, và Yahoo! đều dùng MySQL cho mục đích lưu trữ dữ liệu. Kể cả khi ban đầu nó chỉ được dùng rất hạn chế nhưng giờ nó đã tương thích với nhiều hạ tầng máy tính quan trọng như Linux, macOS, Microsoft Windows, và Ubuntu.

CHƯƠNG 2: THỰC HIỆN

2.1 Workflow

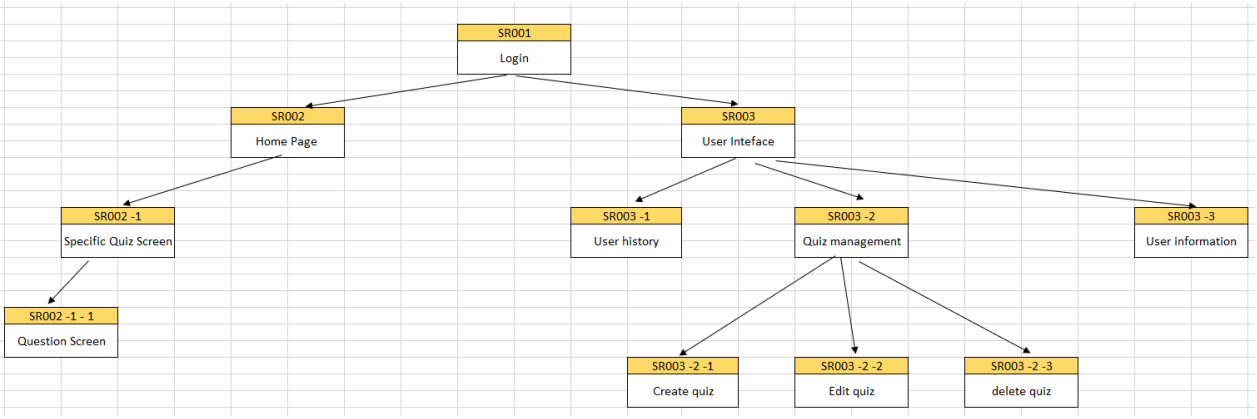
Đầu tiên khi vào màn hình ‘home page’. User sẽ nhìn thấy tất của các quiz có status là public. Khi ấn vào một quiz bất kỳ, người dùng sẽ được chuyển hướng đến màn hình ‘play quiz’. Ở đây người dùng sẽ nhìn thấy được thông tin chi tiết của quiz đó bao gồm tên quiz, mô tả, người tạo, rating, … Người dùng còn có thể thấy được bảng lịch sử thông tin người dùng và số điểm đã chơi quiz này. Ở màn hình ‘play quiz’, người dùng ấn vào nút ‘play quiz’ sẽ được chuyển tới màn hình ‘play quiz’. Ở màn hình này, các câu hỏi sẽ hiển thị lên, cùng với đó là các đáp án, thanh thời gian. Khi người dùng chọn đáp án, tùy vào đáp án đúng hay sai mà số điểm sẽ được cộng vào. Số điểm sẽ bằng số thời gian còn lại nhân cho một trăm. Khi kết thúc lượt chơi. Người dùng sẽ được chuyển đến màn hình ‘finished’. Ở màn hình ‘finished’, hiển thị ra số câu hỏi đúng, số điểm đạt được cũng với đó là cho phép người dùng đánh giá quiz. Kết thúc luồng chạy ‘play quiz’.

Ở mà hình ‘home page’. Người dùng đăng nhập, sau khi đăng nhập xong, ở phần sidebar sẽ hiển thị phần quản lý quiz. Khi người đùng ấn vào nút ‘my quiz’, sẽ chuyển hướng đến màn hình ‘my quiz’. Ở màn hình này, sẽ hiển thị thông tin người dùng, cùng với đó là một bảng các quiz mà người dùng đã tạo. Trên bảng này sẽ có những hành động cho phép người dùng thao tác như tạo một quiz mới, chỉnh sửa quiz đã tạo.

Ở màn hình tạo quiz, người dùng nhập các thông tin cơ bản bao gồm hình ảnh, tên, mô tả, ngôn ngữ, thể loại và thời gian. Sau khi tạo quiz xong, sẽ được chuyển hướng đến màn hình chỉnh sửa. Ở màn hình chỉnh sửa quiz, người dùng có thể chuyển hướng sang màn hình tạo các câu hỏi và câu trả lời. Ở màn hình chỉnh sửa câu hỏi, sẽ hiển thị ra bảng chứa thông tin các câu hỏi và câu trả lời, các nút nhấn để thêm câu hỏi, câu trả lời, nút thêm theo định dạng csv. Khi muốn chỉnh sửa câu hỏi hoặc câu trả lời, người dùng chỉ ấn vào câu hỏi và chỉnh sửa.

2.2 Backend

Đây là các màn hình.



Hình 2.1. Các màn hình

Ứng dụng có tổng cộng khoảng 43 API phục vụ cho việc giao tiếp với máy chủ. Các API dưới đây là những mô tả ngắn gọn về các API này:

Tạo Category:

- Phương thức: POST
- Url: /api/categories

Lấy tất cả category:

- Phương thức: GET
- Url: api/categories

Lấy category theo id:

- Phương thức: GET
- Url: /api/categories/:id
- Mô tả: trả về id, tên của category, cùng với một dãy quiz của category này.

Sửa một category:

- Phương thức: PUT
- Url: /api/categories/:id
- Mô tả: trả về đối tượng đã được chỉnh sửa. Ngược lại quăng lỗi.

Xóa Category:

- Phương thức: DELETE
- Url: /api/categories/:id
- Mô tả: trả về đoạn text khi xóa thành công. Ngược lại báo lỗi.

Tạo Language:

- Phương thức: POST
- Url: /api/languages/
- Mô tả: trả về một đối tượng khi tạo thành công. Ngược lại báo lỗi

Sửa Language:

- Phương thức: PUT
- Url: /api/languages/:id
- Mô tả: Trả về một đối tượng đã được chỉnh sửa. Ngược lại báo lỗi.

Xóa Language:

- Phương thức: DELETE
- Url: api/timings/:id
- Mô tả: Trả về đoạn text nếu xóa thành công. Ngược lại quăng lỗi.

Xem các Language:

- Phương thức: GET
- Url: /api/languages/
- Mô tả: Trả về một dãy cái đối tượng.

Xem Language bằng id:

- Phương thức: GET
- Url: api/languages/:id
- Mô tả: Trả về thông tin language và dãy quiz của language này.

Tạo Timing:

- Phương thức: POST
- Url: api/timing/
- Mô tả: Trả về đối tương nếu được tạo thành công. Ngược lại quăng lỗi.

Chỉnh sửa Timing:

- Phương thức: PUT
- Url: api/timings/:id
- Mô tả: Trả về đối tương đã được chỉnh sủa thành công. Ngược lại quăng lỗi.

Xóa Timing:

- Phương thức: DELETE
- Url: api/timings/:id
- Mô tả: Trả về đoạn text nếu xóa thành công. Ngược lại quăng lỗi.

Xem các Timing:

- Phương thức: GET
- Url: /api/timings/
- Mô tả: Trả về một dãy cái đối tượng.

Xem Timing bằng id:

- Phương thức: GET
- Url: api/timings/:id
- Mô tả: Trả về thông tin timings và dãy quiz của language này.

Xem tất cả User:

- Phương thức: GET
- Url: /api/users
- Mô tả: Trả về tất cả các user có trong hệ thống.

Xem user bằng Id

- Phương thức: GET
- Url: api/users/:id
- Mô tả: Trả về user tương ứng với id.

Thêm một user

- Phương thức: POST
- Url: api/users/
- Mô tả: Trả về đối tượng nếu tạo thành công, ngược lại báo lỗi.

Sửa một user

- Phương thức: PUT
- Url: api/users/:id
- Mô tả: Trả về đối tượng nếu sửa thành công, ngược lại báo lỗi.

Xóa một user

- Phương thức: DELETE
- Url: api/users/:id

- Mô tả: Trả về đoạn text nếu xóa thành công, ngược lại báo lỗi.

Xem tất cả các quiz

- Phương thức: GET
- Url: api/quizzes/
- Mô tả: Trả về tất cả các quiz có state là public.

Xem quiz bằng id

- Phương thức: GET
- Url: api/quizzes/:id
- Mô tả: Trả về quiz có id tương ứng.

Tạo một quiz

- Phương thức: POST
- Url: api/quizzes/
- Mô tả: Trả về đối tương ghi tạo thành công, ngươc lại báo lỗi.

Chỉnh sửa một quiz

- Phương thức: PUT
- Url: api/quizzes/:id
- Mô tả: Trả về đối tượng nếu chỉnh sửa thành công, ngược lại báo lỗi.

Xóa một quiz

- Phương thức: DELETE
- Url: api/quizzes/:id
- Mô tả: Trả về đoạn text nếu xóa thành công, ngược lại báo lỗi.

Tạo question

- Phương thức: POST
- Url: api/questions/:id
- Mô tả: Trả về quiz tương ứng của question được thêm, ngược lại báo lỗi.

Xóa question

- Phương thức: DELETE
- Url: api/questions/:id
- Mô tả: Trả về đoạn text nếu xóa thành công, ngược lại báo lỗi.

Xóa choice

- Phương thức: DELETE
- Url: api/choices/:id
- Mô tả: Trả về đoạn text nếu xóa thành công, ngược lại báo lỗi.

Thêm rating

- Phương thức: POST
- Url: api/ratings
- Mô tả: Trả về đối tượng nếu chỉnh sửa thành công, ngược lại báo lỗi.

Xem history bằng user id

- Phương thức: GET
- Url: api/histories/

- Mô tả: Trả về các history dựa trên id của user.

Xem history bằng ngày tháng cụ thể:

- Phương thức: GET
- Url: api/histories/d
- Mô tả: Trả về các history đựa trên ngày tháng cụ thể.

Thêm history

- Phương thức: POST
- Url: api/histories
- Mô tả: Trả về quiz nếu thêm thành công history. Ngược lại báo lỗi.

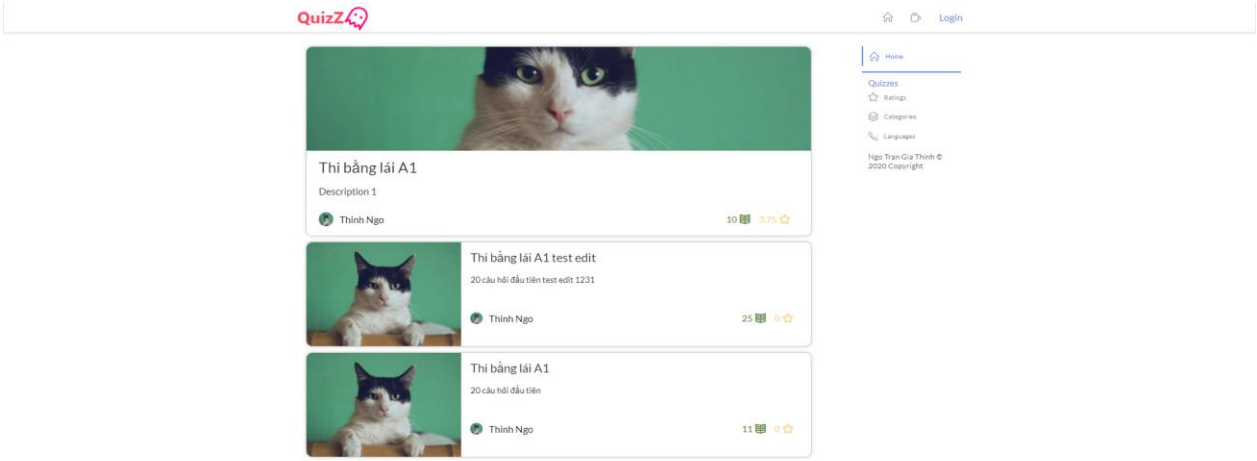
Đăng nhập:

- Phương thức: POST
- Url: api/login
- Mô tả: Trả về một token nếu người dùng đăng nhập thành công. Token sẽ được lưu vào Redis trên máy chủ và Cookies bên phía người dùng.

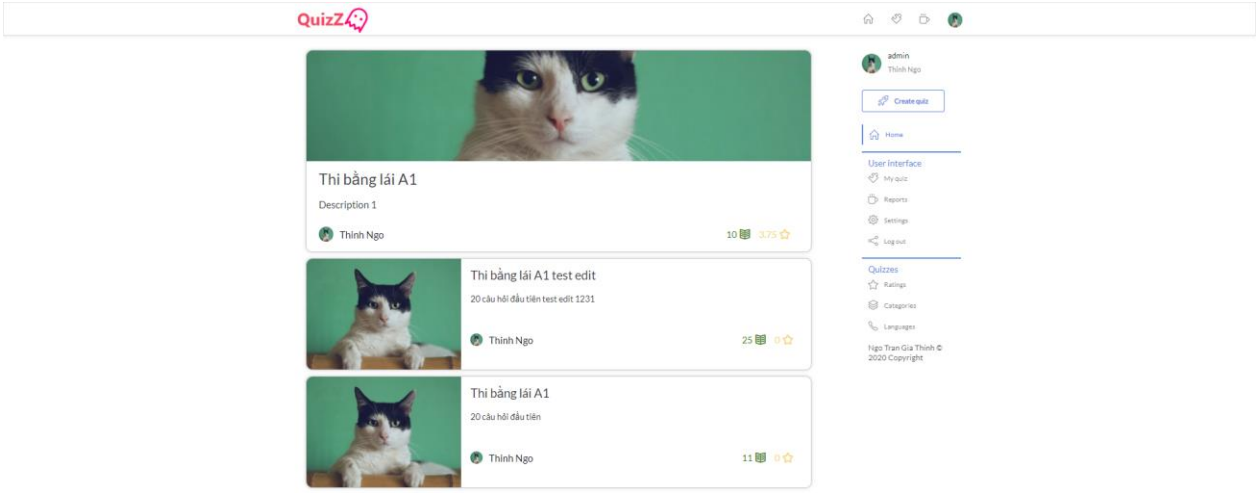
2.3 Frontend

Hệ thống sử dụng ReactJS để làm front-end. Bao gồm khoảng 28 components phục vụ cho việc hiển thị. Các component đó bao gồm:

Homepage: Đây là component chính xuất hiện đầu tiên khi truy cập vào trang web. Chức năng chính của màn hình này là hiển thị tất cả các quiz có trong cơ sở dữ liệu. Các quiz được hiển thị khi state của nó ở trạng thái public. Bên trong component này sẽ bao gồm các component con. Đó là Section, Navigation, Sidebar. Section là component chứa các quiz. Bên trong Section sẽ có hai component con là nhiệm vụ hiển thị thông tin của quiz. Đó là BigCard và MediumCard. Component Navigation dùng để hiển thị thanh chỉ dẫn của toàn trang. Chức năng chính của nó là chỉ hướng đến một trang nào khác thông qua hành động cụ thể của người dùng. Component Sidebar cũng giống như Navigation chỉ hướng đến các màn hình cụ thể thông qua hành động người dùng.



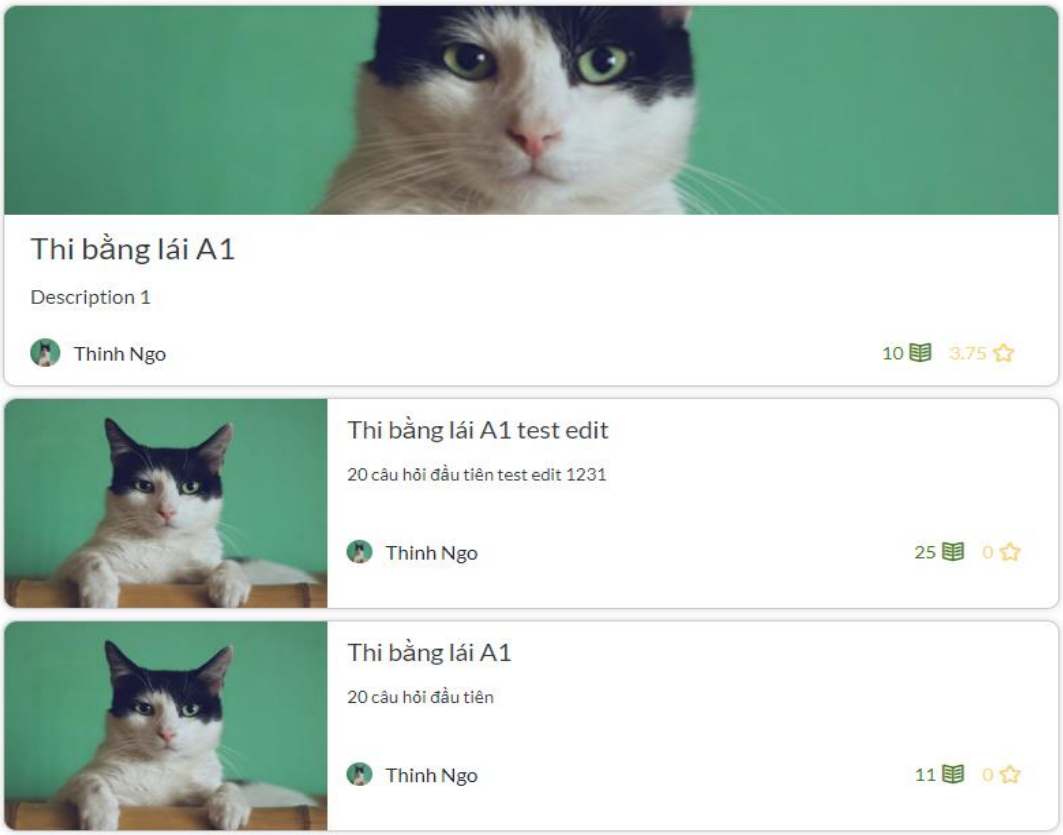
Hình 2.2. Màn hình trang chủ



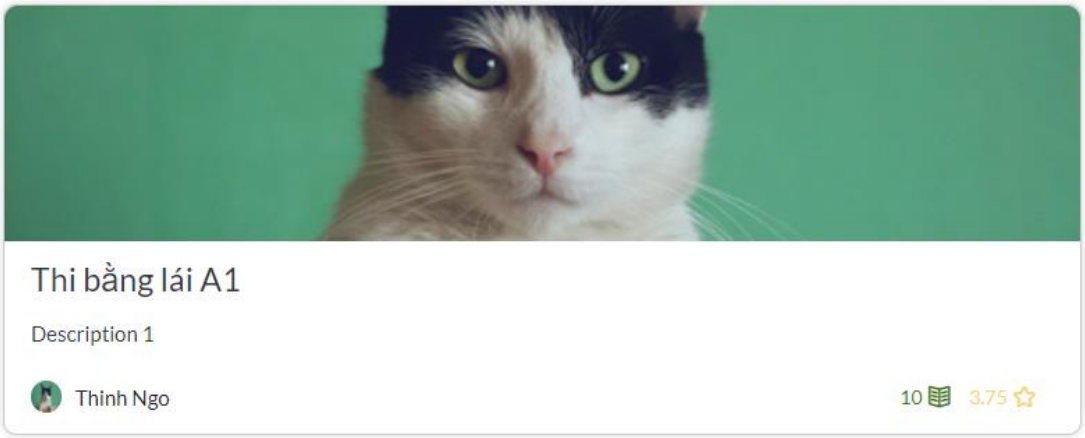
Hình 2.3. Màn hình trang chủ sau khi đăng nhập

Hình 2.4. Thanh chỉ dẫn

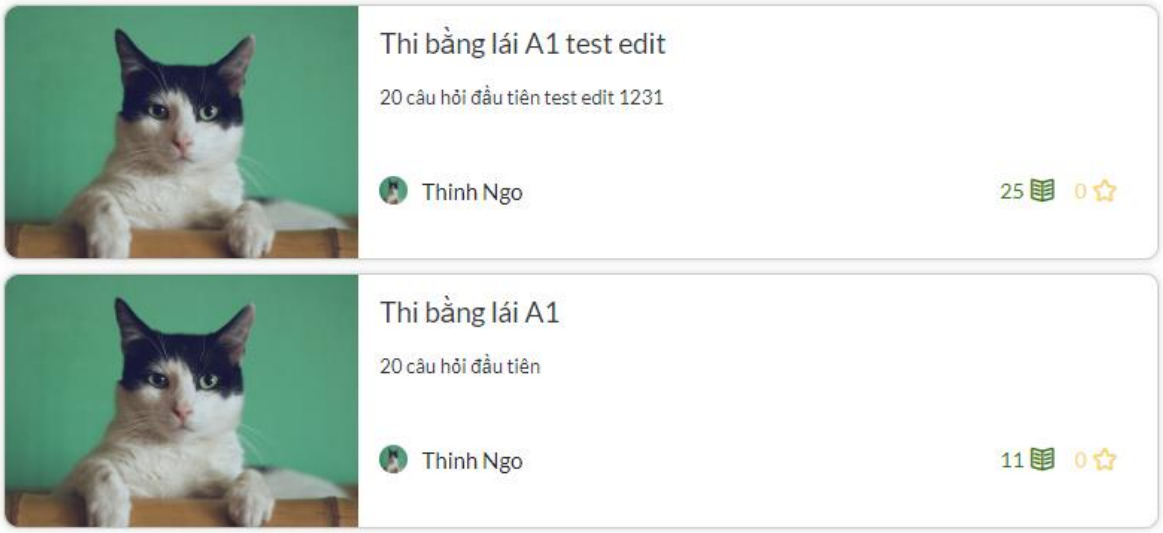
Hình 2.5. Thanh chỉ dẫn sau khi đăng nhập



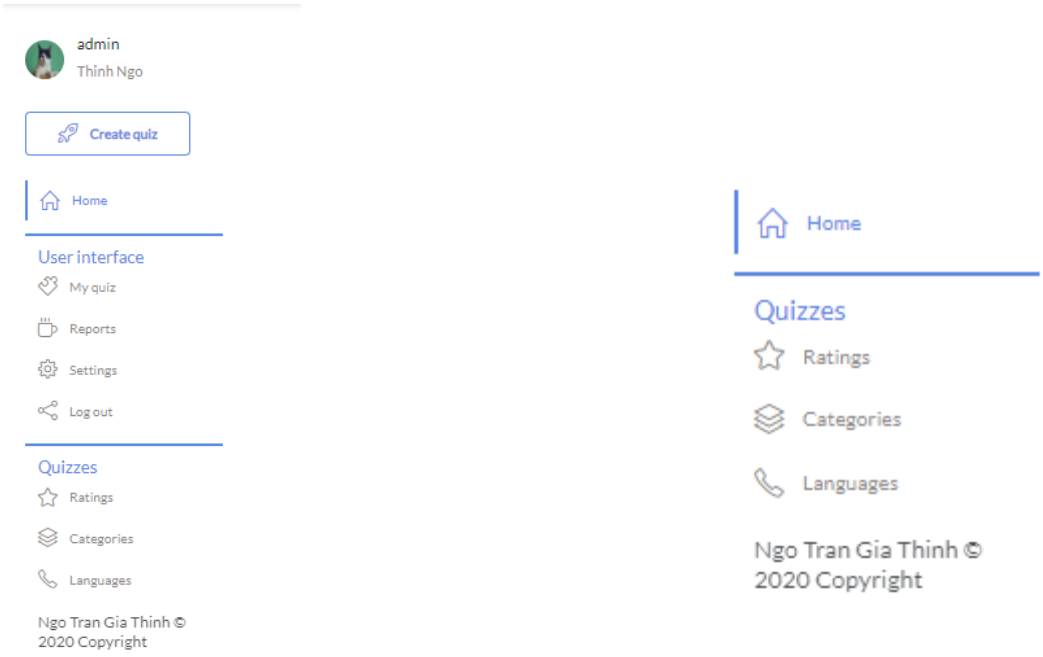
Hình 2.6. Màn hình Section



Hình 2.7. Thẻ lớn



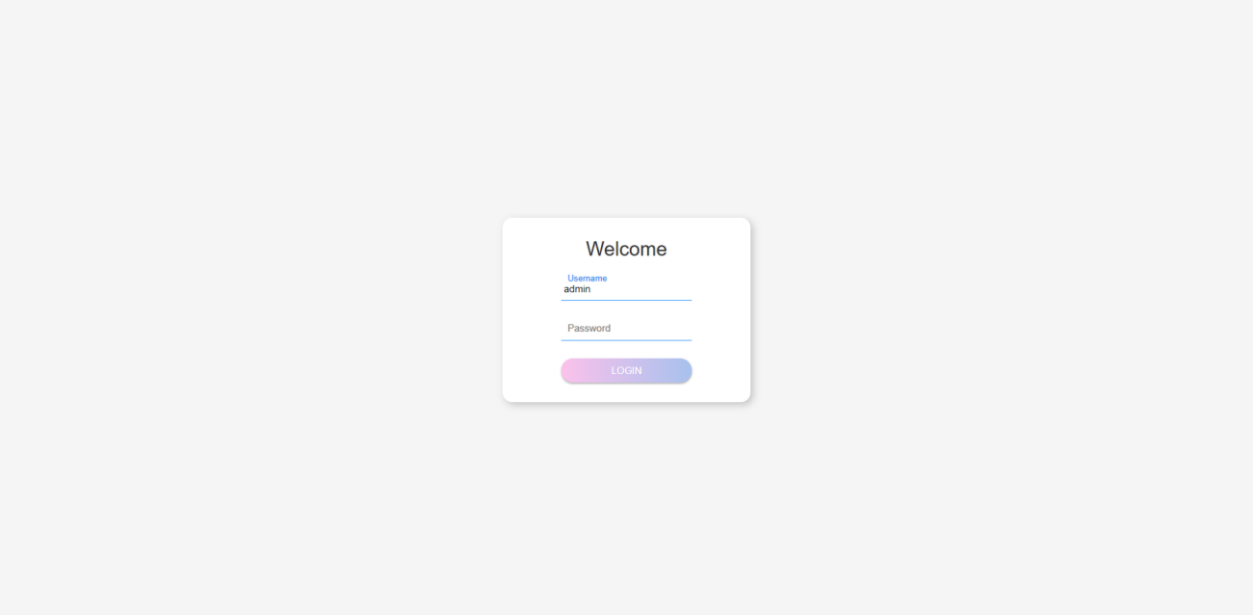
Hình 2.8. Thẻ trung



Hình 2.9. Thanh chỉ dẫn

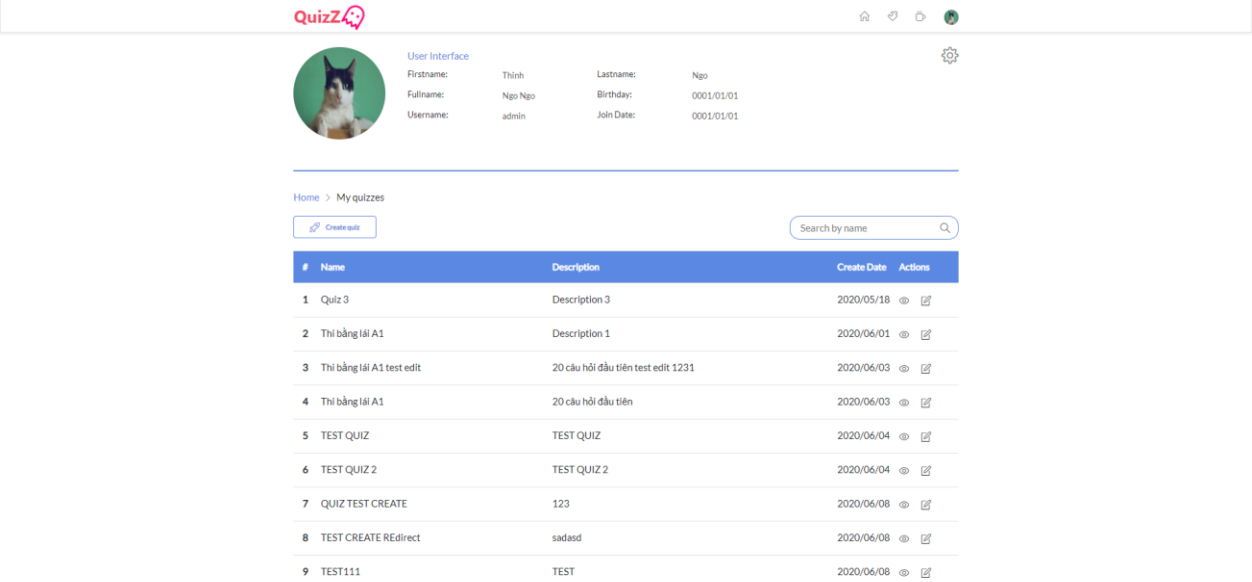
Login: Ở màn hình HomePage, khi người dùng nhấn vào login trên thanh chỉ dẫn. Sẽ được chuyển tới màn hình này. Chức năng chính của màn hình login là để định danh người đùng thông qua đăng nhập. Khi xác thực người dùng thành công máy chủ sẽ tự động tạo ra token và gửi lại cho người dùng. Token này sẽ được lưu vào Redis bên phía

máy chủ và Cookies bên phía người dùng. Nó sẽ được gửi cùng mỗi khi người dùng gửi yêu cầu về phía máy chủ. Token được lưu trong Redis có thời gian là một tuần. Tuy vậy khi người dùng đăng xuất hoặc thay đổi mật khẩu, thì token đó được mặc định là hết hạn.

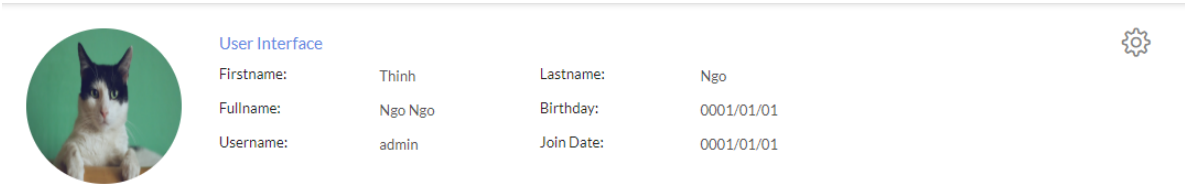


Hình 2.10. Màn hình đăng nhập

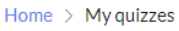
MyQuiz: Chức năng chính của màn hình này là để quản lý tất cả các quiz được người dùng tạo ra. Các component xuất hiện trong màn hình này gồm Navigation, UserInfo, Breadcrumbs. UserInfo được dùng để hiển thị thông tin người dùng. Thông tin người dùng sẽ được lấy thông qua Token chứa trong Cookies. Breadcrumbs được dùng để hiển thị dấu trang mà người dùng đi qua, chức năng chính là giúp người dùng có thể quay lại trang trước đó. Dưới Breadcrumbs là một bảng hiển thị các quiz. Ở bảng này sẽ hiển thị thông tin cơ bản của quiz đó. Nó có các hành động để dẫn sang màn hình chỉnh sửa quiz hoặc chơi quiz ở bảng này.



Hình 2.11. Màn hình quản lý quiz

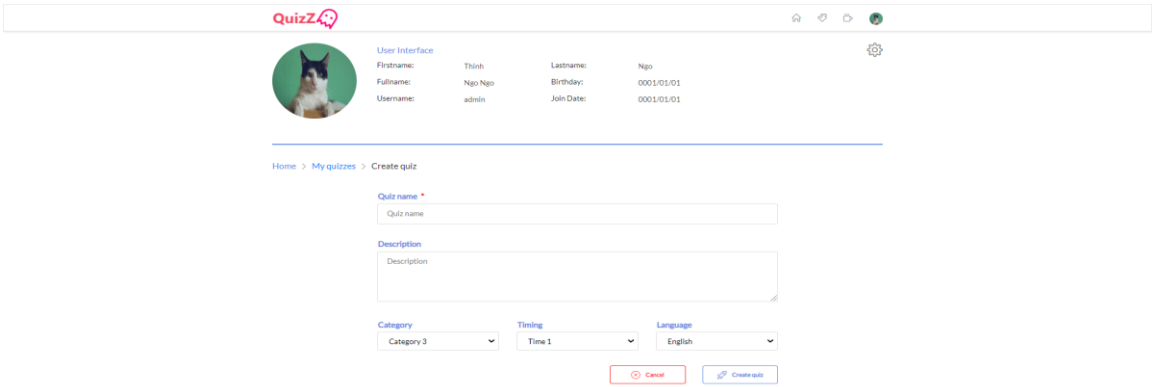


Hình 2.12. Thông tin người dùng



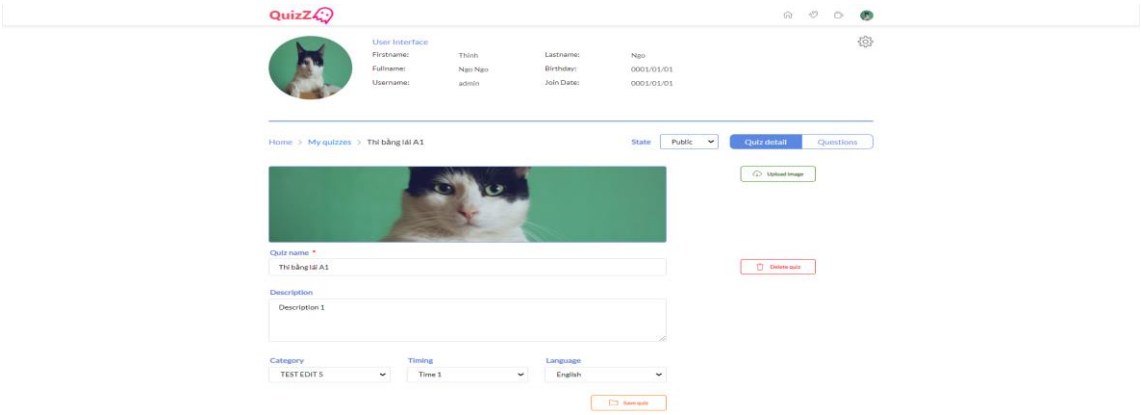
Hình 2.13. Mục chỉ dẫn

CreateQuiz: Chức năng ở màn hình này là giúp người dùng tạo quiz. Sau khi nhập đầy đủ thông tin. Hệ thống sẽ kiểm tra tên của quiz có bị trùng hay không. Nếu có sẽ báo lỗi, nếu không hệ thống sẽ ghi nhận và chuyển tiếp màn hình chỉnh sửa quiz.



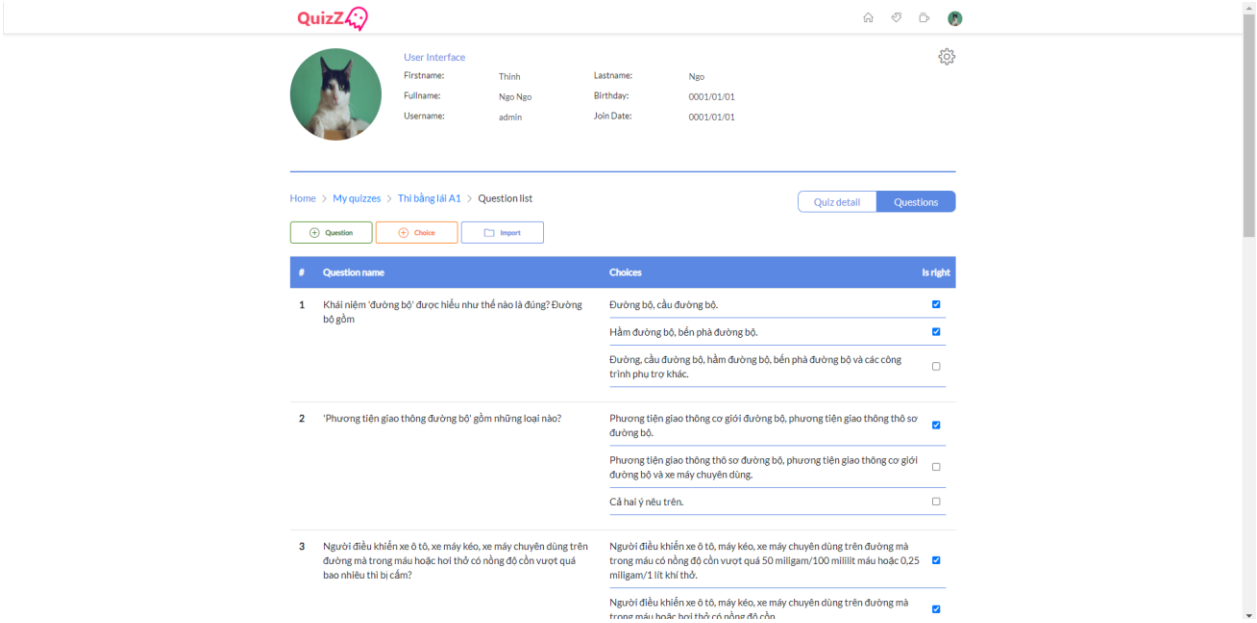
Hình 2.14. Màn hình tạo quiz

EditQuiz: Màn hình chỉnh sửa quiz, chức năng của màn hình này là cho người dùng chỉnh sửa quiz hiện có. Ở màn hình này, có thể thêm được hình ảnh, chỉnh sửa state cho quiz. Nếu state có hai loại đó là Public và Private, Nếu public thì quiz này sẽ được hiển thị trên màn hình HomePage và ngược lại.

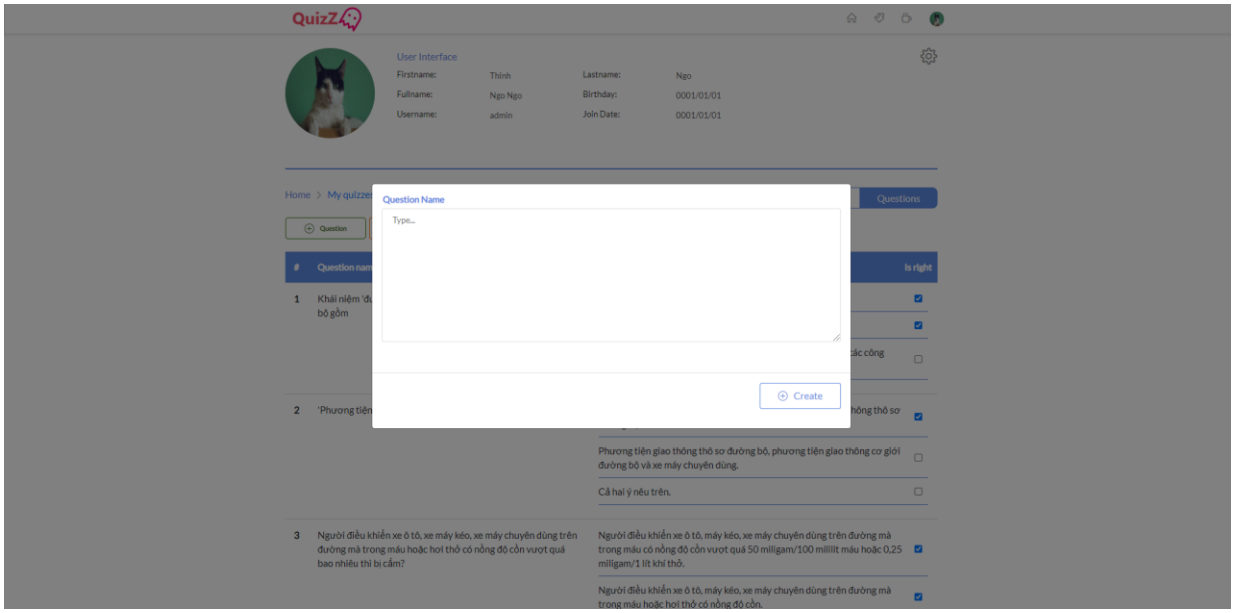


Hình 2.15. Màn hình chỉnh sửa quiz

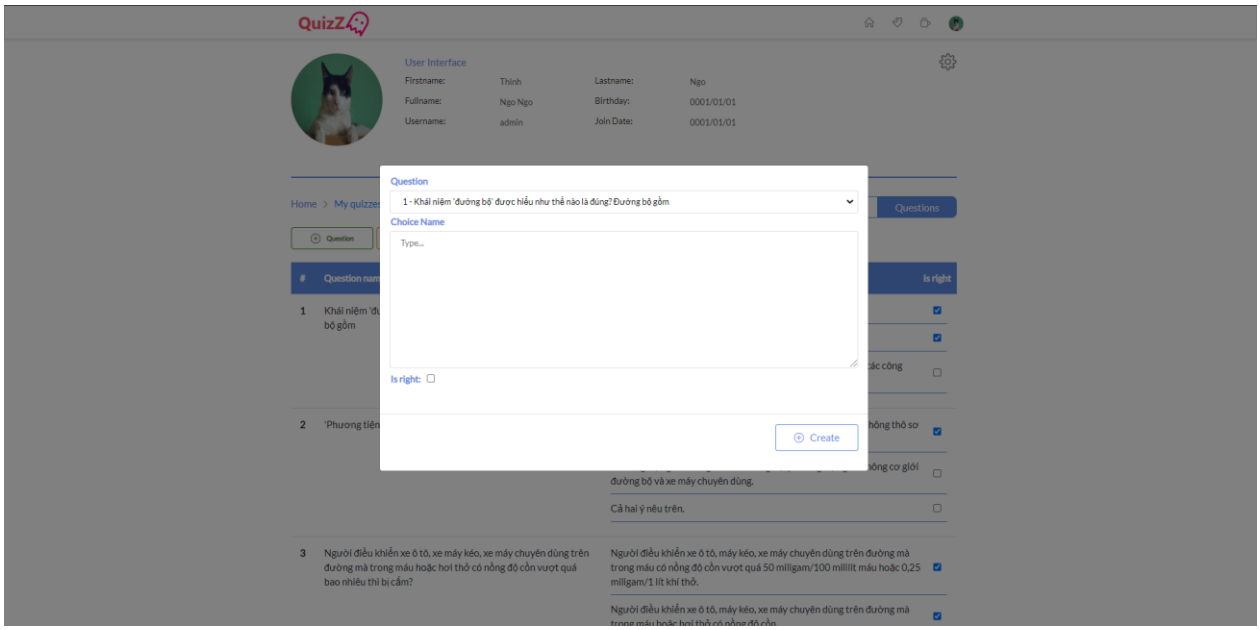
EditQuestion: Ở màn hình chỉnh sửa quiz, khi ấn nút Questions thì sẽ được chuyển tiếp đến màn hình này. Ở màn hình chỉnh sửa câu hỏi, sẽ hiển thị ra một bảng tổng hợp tất cả các câu hỏi và đáp án có trong quiz. Người dùng có thể thao tác trên bảng này bằng cách double-click vào từng trường tương ứng. Lập tức sẽ đươc chuyển sang chế độ chỉnh sửa. Người dùng có thể thêm câu hỏi bằng cách án vào nút Question, thêm đáp án cho câu hỏi bằng cách ấn vào nút Choice. Nếu muốn thêm nhanh một loạt các câu hỏi và đáp án, thì người dùng thêm bằng cách ấn vào nút Import, ở đây sẽ thêm theo đinh dạng csv hoặc excel. Câu hỏi và đáp án sẽ phân cách với nhau bằng đấu cách, xuống dòng là kết thúc một câu hỏi.



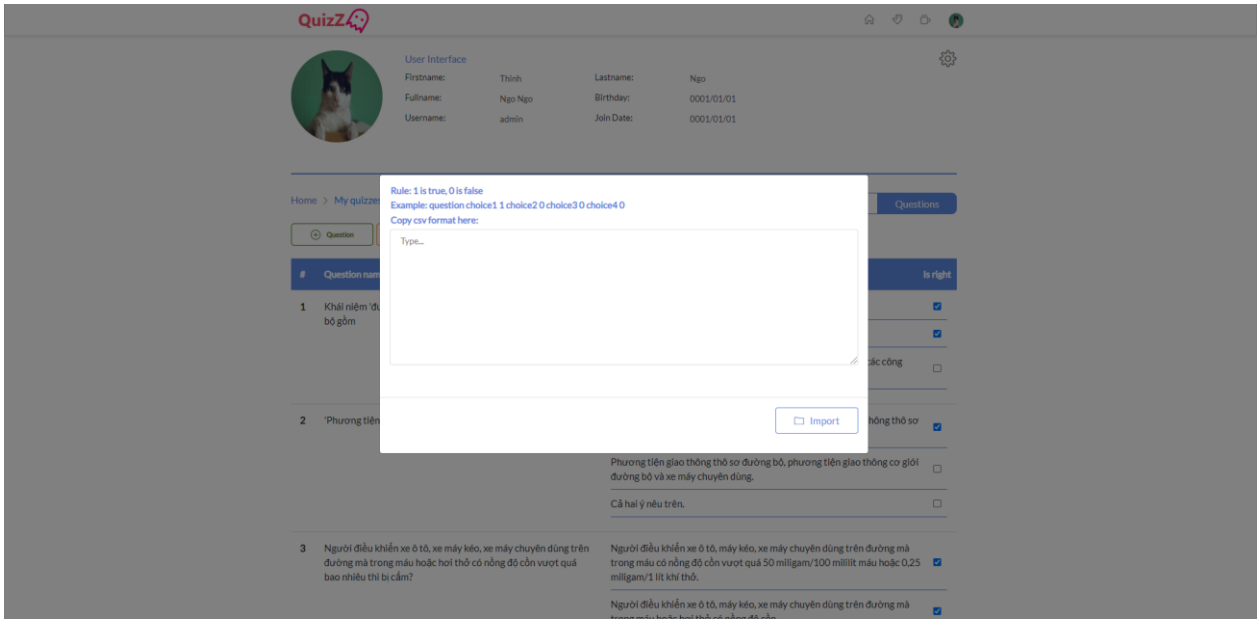
Hình 2.16. Màn hình chỉnh sửa câu hỏi



Hình 2.17. Màn hình thêm câu hỏi

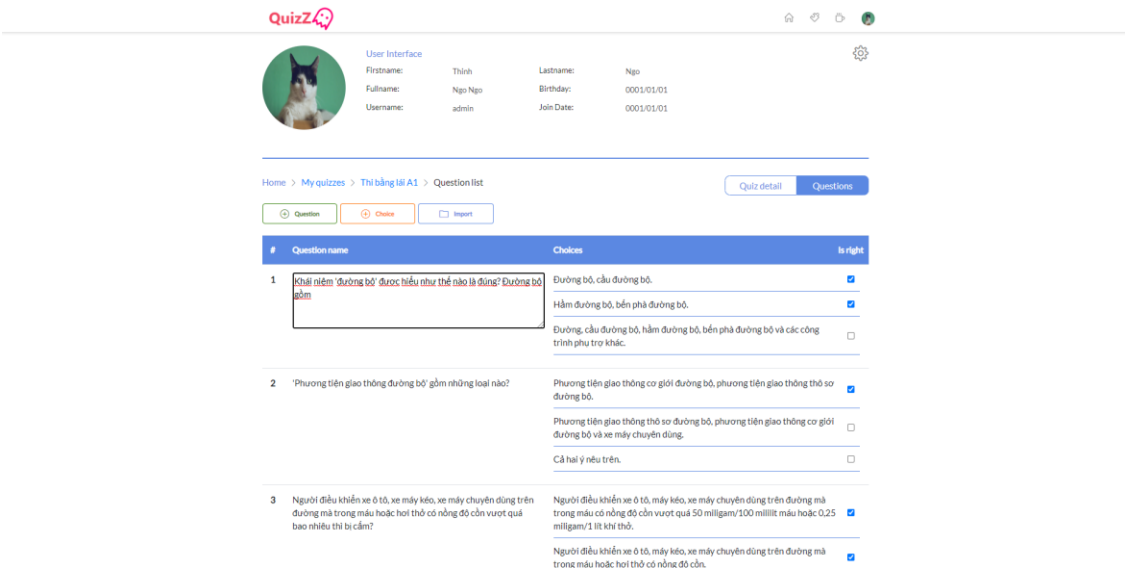


Hình 2.18. Màn hình thêm đáp án

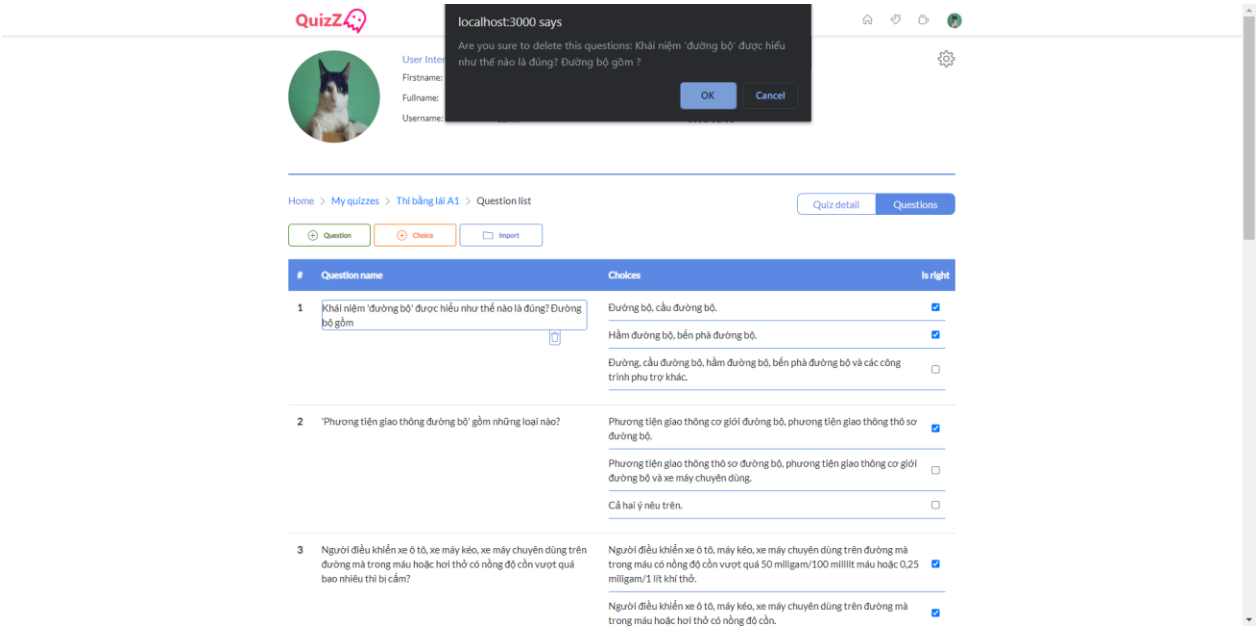


Hình 2.19. Màn hình thêm theo định dạng của csv hoặc excel

Khi muốn chỉnh sửa chỉ cần double-click vào text tương ứng.

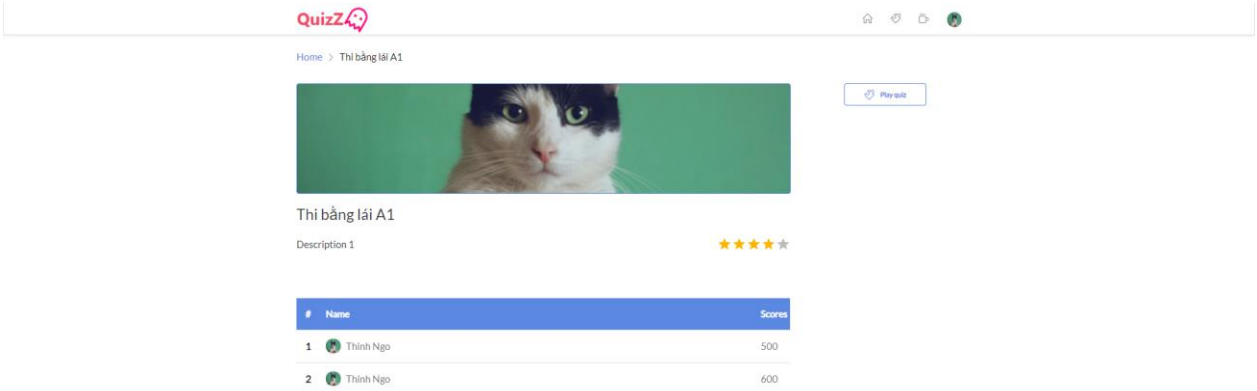


Hình 2.20. Màn hình chỉnh sửa câu hỏi hoặc đáp án bất kỳ

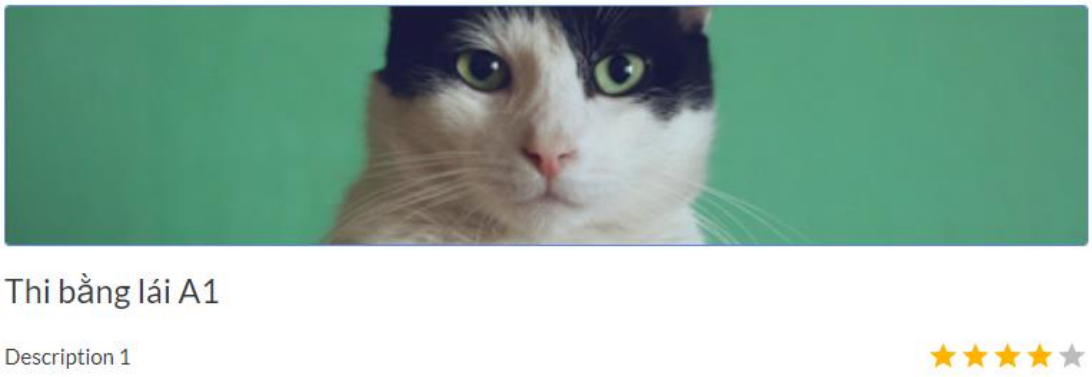


Hình 2.21. Màn hình xóa câu hỏi hoặc đáp án bất kỳ

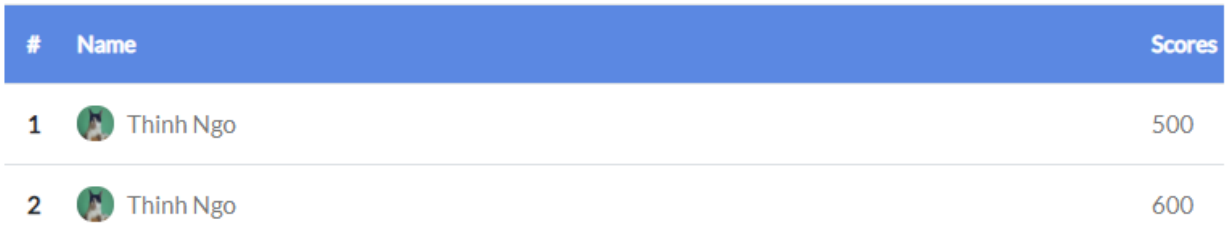
QuizDetail: Màn hình chi tiết của quiz, ở màn hình này sẽ hiển thị thông tin chi tiết của quiz. Các component có trong màn hình này là QuizInfo, HistoryTable. QuizInfo dùng để hiển thị thông tin của quiz. HistoryTable được dùng để hiển thị lịch sử người dùng khác đã chơi quiz này.



Hình 2.22. Màn hình chi tiết quiz

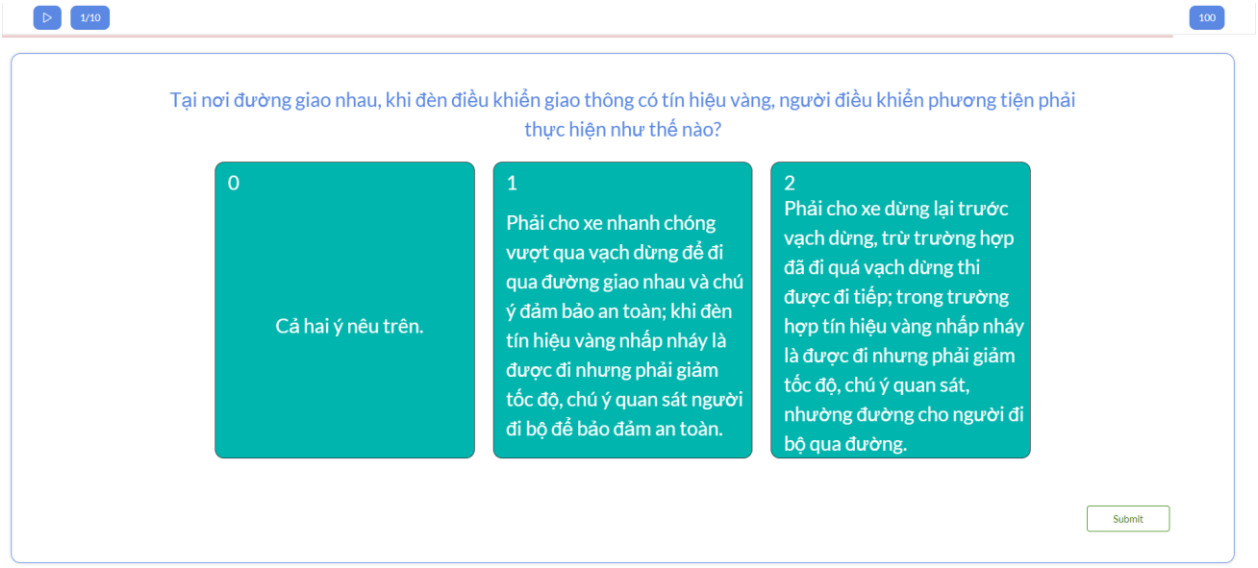


Hình 2.23. Thông tin quiz



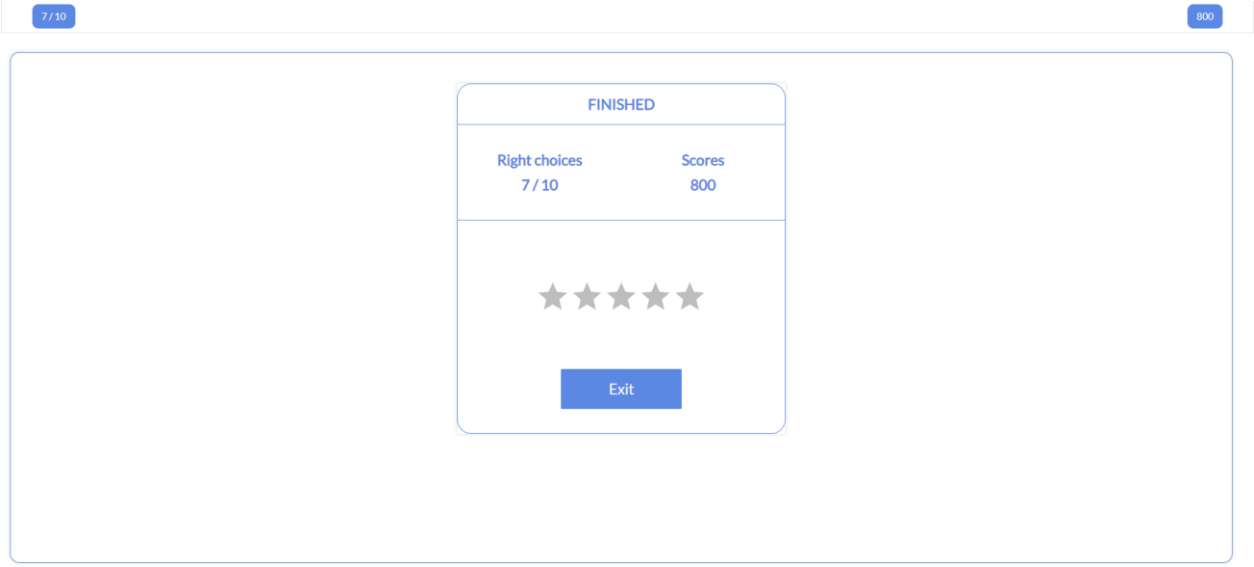
Hình 2.24. Bảng lịch sử

PlayQuiz: Khi người dùng ấn vào nút Play Quiz trong màn hình QuizDetail, sẽ được chuyển hướng sang màn hình này. Ở màn hình PlayQuiz sẽ hiển thị lần lượt câu hỏi và các đáp án. Nhiệm vụ của người dùng là lựa chọn đáp án trong khoảng thời gian cho phép sau đó ấn vào nút submit để chốt đáp án, sau khi chốt đáp án màn hình sẽ hiển thị các đáp án đúng. Người đùng có thể chọn nhiều đáp án trong mỗi câu hỏi. Các câu hỏi và đáp án sẽ được xáo trộn ngẫu nhiên. Mỗi câu trả lời đúng sẽ được cộng vào một trăm điểm. Cuối cùng sau khi hoàn thành trò chơi, người đùng sẽ được chuyển đến màn hình FinshQuiz.

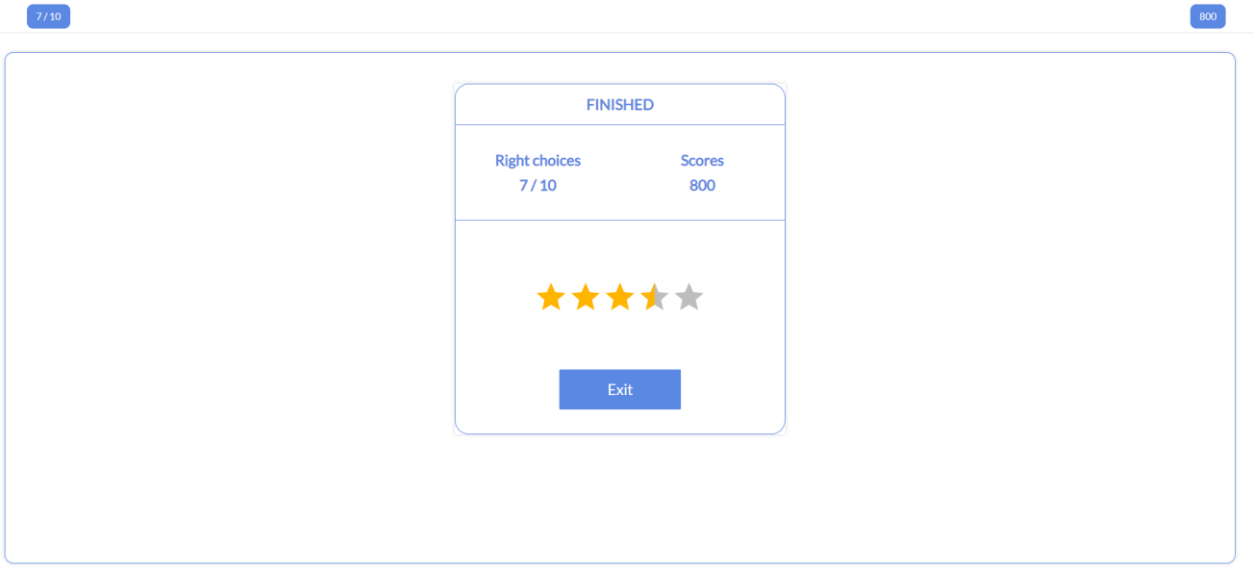


Hình 2.25. Màn hình chơi các quiz

FinshQuiz: Chức năng chính của màn hình này hiển thị thông tin tổng hợp người dùng nhận được bao gồm số điểm và số đáp án đúng. Ở màn hình này, người dùng còn có thể đánh giá cho quiz bằng cách ấn vào các dấu sao. Nếu người dùng ấn vào nút Exit, thì hệ thống sẽ chuyển về màn hình HomePage.



Hình 2.26. Màn hình khi kết thúc quiz



Hình 2.27. Màn hình sau khi đánh giá xong quiz

CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

3.1 Kết Luận

Với việc phân chia sử dụng công nghệ cụ thể cho backend và frontend, xây dựng được bộ khung vững trãi làm tiền đề phát triển, đã đem lại những ưu điểm sau:

- Dễ dàng bảo trì, phát triển cũng như nâng cấp các tính năng mới thông qua việc phân chia độc lập công việc giữa server và client.
- Tìm hiểu và sử dụng được các công nghệ mới qua đó tối ưu được hiệu xuất cũng như là làm tăng trải nghiệm người dùng.
- Hiểu được quy trình phát triển phần mềm, xậy đựng được hệ thống cơ sở dữ liệu giúp lưu trữ được thông tin.
- Biết sử dụng các thư viện của bên thứ ba. Phục vụ tốt cho việc phát triển ứng dụng qua đó tiết kiệm thời gian và công sức.

Tuy nhiên, bên cạch những ưu điểm trên. Còn có những nhược điểm chưa thể khắc phục hoàn chỉnh:

- Đồng bộ dữ liệu giữa phía server và client đôi khi bị bất đồng bộ.
- Mặc dù đã thêm được hiệu ứng nhưng chuyển động vẫn chưa được mượt mà như mong đợi.
- Việc khai thác điểm mạnh của công nghệ sử dụng vào ứng đụng chưa thực sự hiệu quả như mong đợi.

3.2 Hướng Phát Triển.

Mục tiêu của hướng phát triển này là mở rông và nâng cấp ứng dụng, giải quyết các vấn đề còn tồn tại trong ứng dụng. Ứng dụng đã xây dựng được bộ khung cơ bản nên việc phát triển thêm tính năng cũng sẽ trở nên dễ dàng hơn. Các tính năng mong muốn phát triển là:

- Thêm ngẫu nhiên cách hiển thị câu hỏi và câu trả lời (ví dụ hiển thị cho phép người chơi nhập câu trả lời mà không cần phải chọn, cho phép người chơi nghe một đoạn hội thoại ngắn rồi điền đáp án,…).
- Thêm tính năng gợi ý câu trả lời bằng cách điền thêm thông tin gợi ý vào câu hỏi.
- Thêm tính năng cho phép bình luân và viết cảm nghĩ về một quiz cụ thể sau khi chơi xong quiz đó.
- Thêm màn hình tổng hợp tất cả các hoạt động người dùng để thống kê người dùng đó thích gì, qua đó có thể đề xuất các quiz giúp trải nghiệp người dùng tốt hơn.

CHƯƠNG 4: TÀI LIỆU THAM KHẢO

[1] Tài liệu Gin Framework
https://github.com/gin-gonic/gin
[2] Tài liệu về ReactJs
https://reactjs.org/
[3] Tài liệu về Gorm
https://gorm.io/
[4] Tài liệu JWT
https://jwt.io/
[5] Tài liệu Redis
https://redis.io/
[6] Tài liệu TypeScript
https://www.typescriptlang.org/
[7] Tài liệu React-router
https://reactrouter.com/web
[8] Tài liệu React-spring
https://www.react-spring.io/
[9] Tài liệu Cookie-universal
https://github.com/microcipcip/cookie-universal/tree/master/packages/cookie-universal#readme
[10] Tài liệu Axios
https://github.com/axios/axios
[11] Tài liệu Sass/Scss
https://sass-lang.com/guide