TÉCNICAS AVANZADAS DE PROGRAMACIÓN

Trabajo Práctico Integrador: Entrega Final

Profesora
MARÍA BELÉN ALEGRE

Alumno
FACUNDO ESTEBAN PALERMO
Legajo 0112674

Carrera
Licenciatura en Informática

Fecha
23 de noviembre de 2022

Contenido

Objetivo

Se requiere realizar un sistema de gestión para la asignación de turno y entrega de registro de conducción.

Requerimientos Funcionales

1. El examen consta de dos etapas. La primera etapa incluye la pregunta si utiliza o no anteojos.
    a. Si responde que no, automáticamente deberá generarse el examen y permitirle responder el cuestionario.
    b. Si responde que sí, se generará un turno random para que asista a una revisión. (La fecha deberá generarse de manera aleatoria).
2. El usuario deberá generar una clave para poder acceder al examen, la cual tendrá un tiempo de expiración de una hora.
3. El cuestionario constará de 10 preguntas precargadas de opción múltiple. Las mismas deberán cargarse de forma aleatoria.
4. La clave generada podrá utilizarse una sola vez. En caso de que no termine la evaluación no podrá volver a usarla.
5. Aprueba el examen con unas 8 preguntas respondidas correctamente.
6. En caso de reprobar el sistema deberá generar un nuevo examen aleatorio.
7. La misma persona sólo podrá rendir un máximo de 3 veces el examen.
8. Los resultados del examen deberán guardarse durante 1 semana.
9. Las preguntas deberán poder ser editables por un administrador del sistema.
10. El administrador del sistema podrá acceder a una estadística diaria de cuantos rindieron el examen, cuantos aprobaron, reprobaron o estuvieron ausentes.

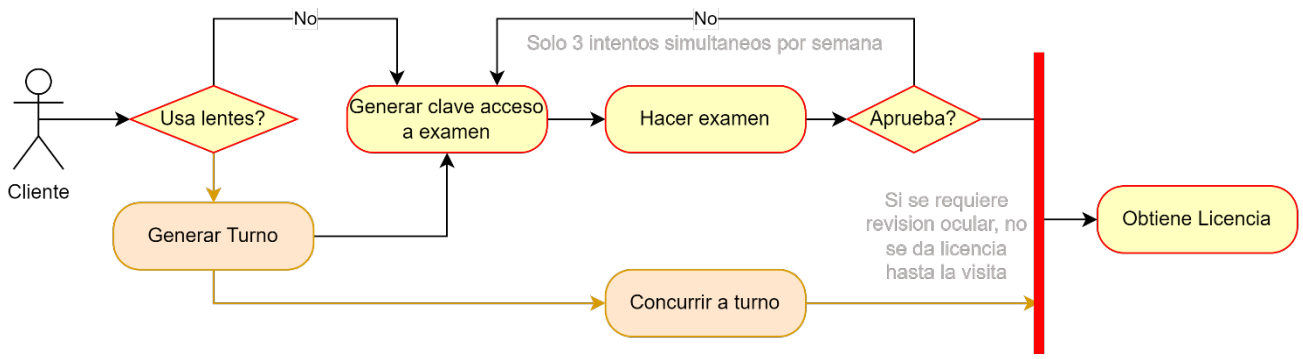
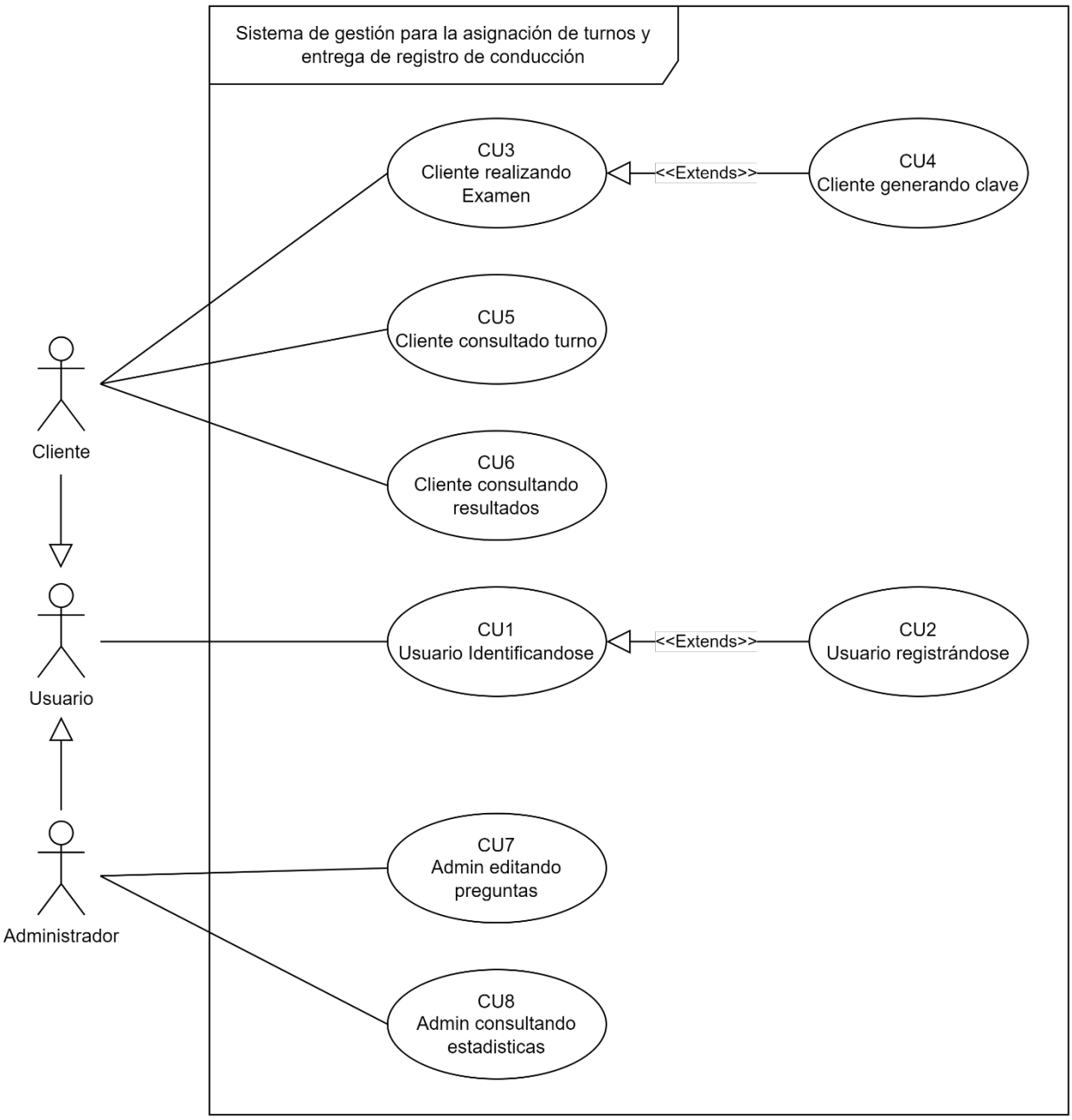


Diagrama demostrativo

Casos de Uso

Diagrama de Casos de Uso



Nombre	CU1. Usuario Identificándose	
Actor	Usuario (P)	
Descripción	Usuario se identifica para obtener acceso a las funciones del sistema limitado al rol que tiene (Cliente o Administrador)	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Usuario ingresa a sección “Iniciar sesión”. El sistema muestra pantalla de inicio de sesión con los labels e inputs de Email y Contraseña y button “Ingresar”. También muestra link para Registrarse.
	2	Usuario ingresa datos en el formulario de inicio de sesión y presiona “Ingresar”. El sistema valida usuario, genera sesión y redirecciona a página principal.
	3	FCU
Postcondición	Usuario identificado con el rol y permisos correspondientes.	
Excepciones	Paso	Acción
	1.1	Usuario no registrado
	1.2	Si [Usuario hace click en Registrarse], ejecutar CU2 Usuario registrándose.
	1.3	FCU
	2.1	Sistema no valida usuario
	2.2	Sistema muestra mensaje de “Usuario o contraseña inválido”
	2.3	Ir a paso 2

Nombre	CU2. Usuario registrándose	
Actor	Usuario (P)	
Descripción	Usuario no registrado adquiere una cuenta	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Usuario ingresa a sección “Registrarse”. El sistema muestra pantalla de registro con los labels e inputs de Nombres, Apellidos, DNI, Dirección, Email, Teléfono, y Contraseña y button “Registrarse”.
	2	Usuario ingresa datos en el formulario de registro y presiona “Registrarse”. El sistema valida datos ingresados, genera usuario y redirecciona a sección de inicio de sesión.
	3	FCU
Postcondición	Usuario registrado con el rol de Cliente.	
Excepciones	Paso	Acción
	2.1	Sistema rechaza registro porque email/DNI, ya existe o datos ingresados incorrectos (DNI, email, contraseña)
	1.2	Sistema muestra mensaje con descripción del error correspondiente.
	1.3	Ir a paso 2

Nombre	CU3. Cliente realizando Examen	
Actor	Cliente (P)	
Descripción	Cliente inicia examen para licencia de conducir	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Cliente registrado ingresa a sección de examen y sistema valida que es primer intento
	2	Sistema muestra pregunta “Debe utilizar anteojos” y botones de “Si, debo utilizar anteojos” y “No, no debo utilizar anteojos”
	3	Cliente selecciona que “No, no debo utilizar anteojos” y Sistema genera un examen
	4	Sistema redirecciona a página de inicio de examen.
	5	Sistema muestra pantalla con campo para ingresar clave de permiso de examen y link para generar clave. Usuario ingresa clave y sistema valida clave.
	6	Sistema muestra examen con 10 preguntas con respuestas multiplechoice ordenadas aleatoriamente de tipo input radio. Al final un button de “Enviar”. Sistema marca clave de acceso como utilizada. Sistema registra 1 intento
	7	Cliente completa cuestionario y hace click en “Enviar”.
	8	Cliente responde al menos 8 preguntas correctamente, y el sistema muestra resultado de Aprobado.
	9	FCU
Postcondición	Cliente ha obtenido una calificación de examen (Aprobado, Desaprobado, Ausente).	
Excepciones	Paso	Acción
	1.1	No es primer intento
	1.2	Ir a paso 4
	2.1	Cliente selecciona “Si, debo utilizar anteojos”, sistema genera un turno aleatorio para revisión e informa a usuario.
	2.2	Ir a paso 3
	4.1	Cliente ya realizo 3 intentos.

	4.2	Sistema muestra mensaje a Cliente “Usted ya ha realizado 3 intentos. Ya no puede realizar el examen. Comuníquese con la oficina de Licencias”.
	4.3	FCU
	5.1	Cliente no ingresa clave
	5.2	Si [Cliente hace click en link de “Generar clave”], ejecutar CU4
	5.3	FCU
	7.1	Cliente no completa cuestionario
	7.2	Sistema registra resultado como Ausente.
	7.3	FCU
	8.1	Cliente responde menos de 8 preguntas correctamente, y el sistema muestra resultado de Desaprobado.
	8.2	FCU

Nombre	CU4. Cliente generando Clave	
Actor	Cliente (P)	
Descripción	Cliente solicita clave de acceso al examen para poder rendirlo.	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Cliente ingresa a sección “Generar clave de acceso a examen”.
	2	El sistema genera una clave alfanumérica aleatoria única de duración de una hora.
	3	El sistema muestra por pantalla la clave generada.
	3	FCU
Postcondición	Cliente obtiene clave de acceso a examen	

Nombre	CU5. Cliente consultando turno	
Actor	Cliente (P)	
Descripción	Cliente consulta el historial de sus turnos para revisión.	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Cliente ingresa a sección “Mis turnos”.
	2	El sistema muestra un listado de los turnos para revisión del cliente con detalle del día y estado (Confirmado / Cancelado) o “Sin turnos otorgados”.
	3	FCU
Postcondición	Cliente consultó sus turnos para revisión.	

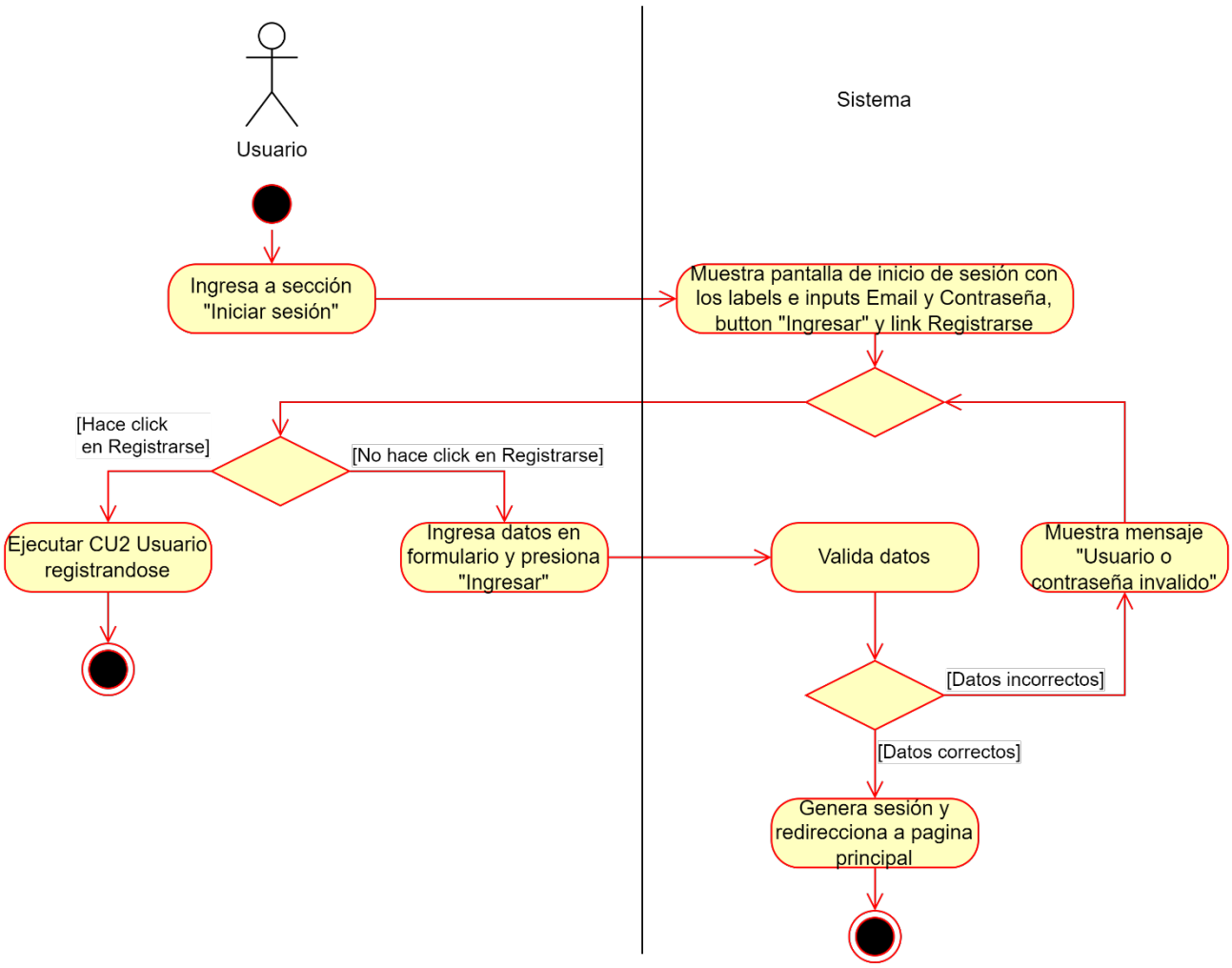
Nombre	CU6. Cliente consultado resultados	
Actor	Cliente (P)	
Descripción	Cliente consulta el historial de los resultados de los exámenes que rindió.	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Cliente ingresa a sección “Mis resultados”.
	2	El sistema muestra un listado de los exámenes que el cliente completó o no, con detalle del día, hora, y resultado (Aprobado, Desaprobado, Ausente) o “Sin exámenes iniciados”.
	3	FCU
Postcondición	Cliente consultó sus resultados de exámenes.	

Nombre	CU7. Admin editando preguntas	
Actor	Administrador (P)	
Descripción	Administrador edita una pregunta a ser utilizadas en los exámenes	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Administrador ingresa a sección “Editar preguntas”.
	2	Sistema despliega en pantalla listado de preguntas con un button de edición para cada una.
	3	Administrador hace click en el button de edición de la pregunta seleccionada.
		Sistema muestra ventana con campo de texto para editar la pregunta, y listado de las respuestas y un campo “id de respuesta correcta” para indicar la respuesta correcta. Junto a las opciones de respuesta hay un button para eliminarlas, y otro button para agregar nueva opción de respuesta. También sistema muestra button de Guardar Pregunta.
	4	Administrador realiza los cambios necesarios y da a guardar Pregunta.
	5	FCU
Postcondición	Administrador ha editado una pregunta.	

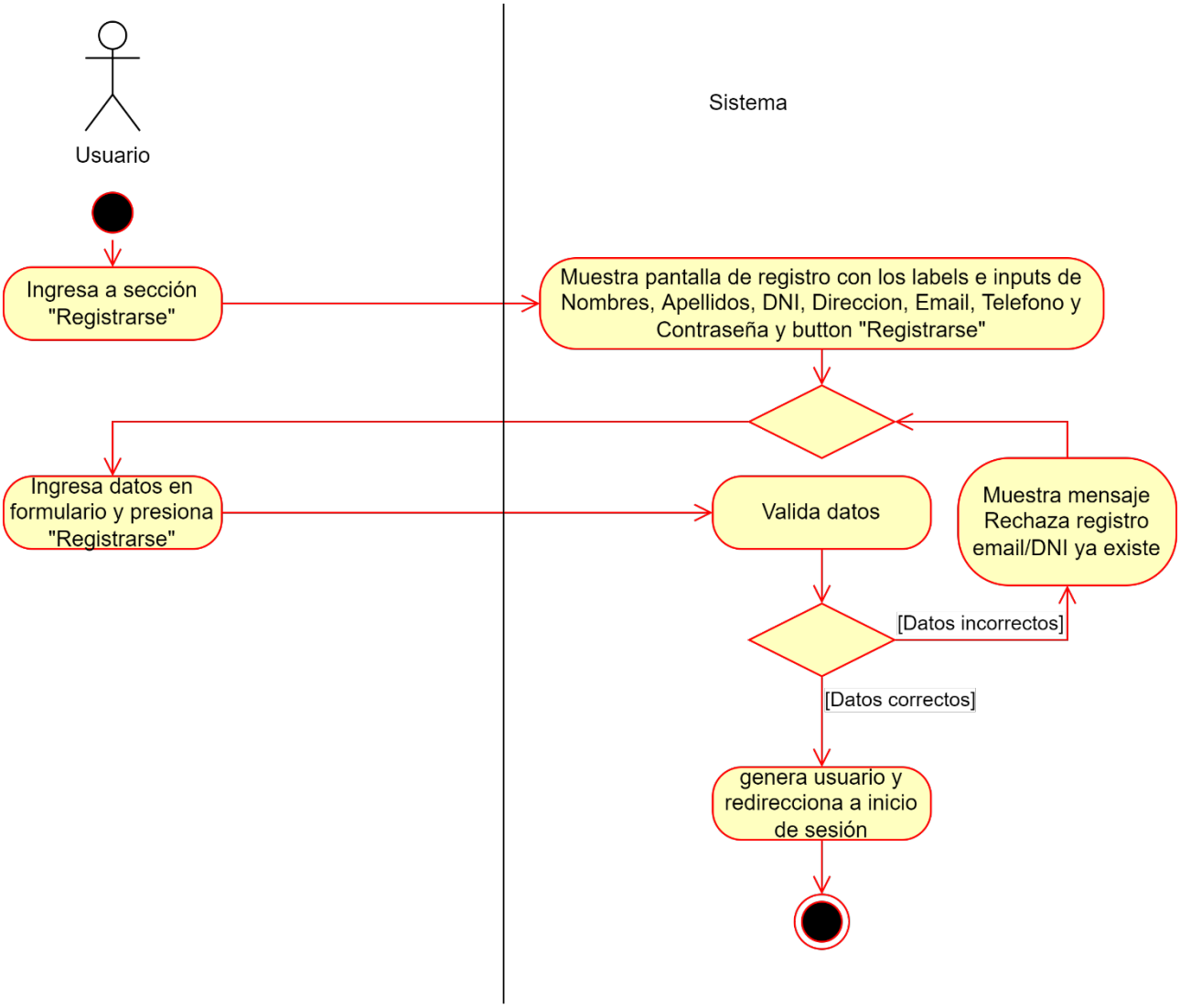
Nombre	CU8. Admin consultando estadísticas	
Actor	Administrador (P)	
Descripción	Administrador requiere información estadística diaria sobre los resultados de los exámenes.	
Precondición	-	
Secuencia Normal	Paso	Acción
	1	Administrador ingresa a sección “Estadísticas”.
	2	Sistema muestra en pantalla un filtro por fecha y button de consultar. A continuación, se muestra resultado para {Hoy} por defecto de fecha, los datos de cantidad de exámenes rendidos, cantidad que aprobó/reprobó/ausentes
	3	Administrador ingresa fecha y hace click en “Consultar” Sistema actualiza datos mostrados para la fecha seleccionada.
	4	FCU
Postcondición	Administrador ha consultado las estadísticas de exámenes.	

Diagramas de actividades

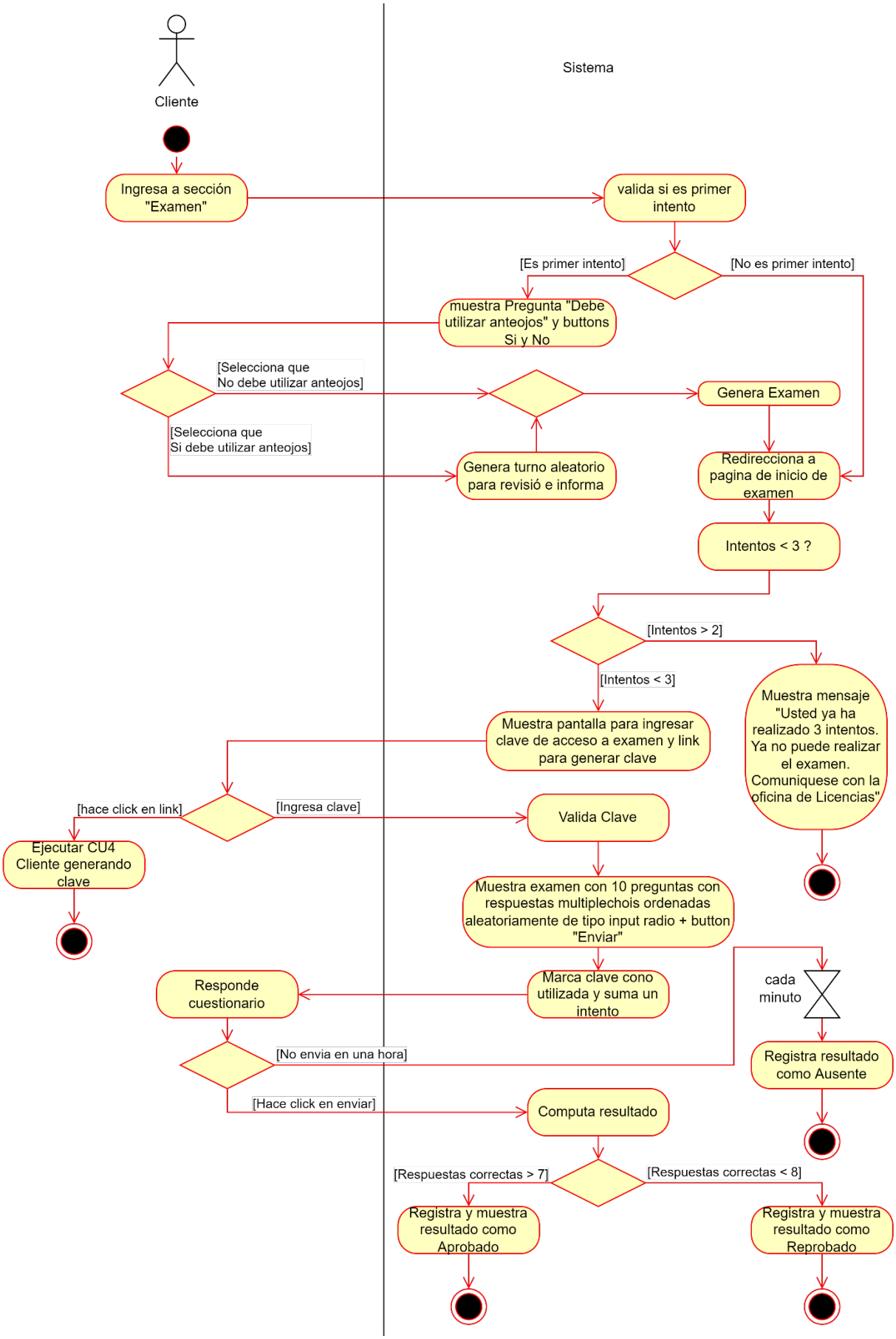
DACU1 – Usuario Identificándose



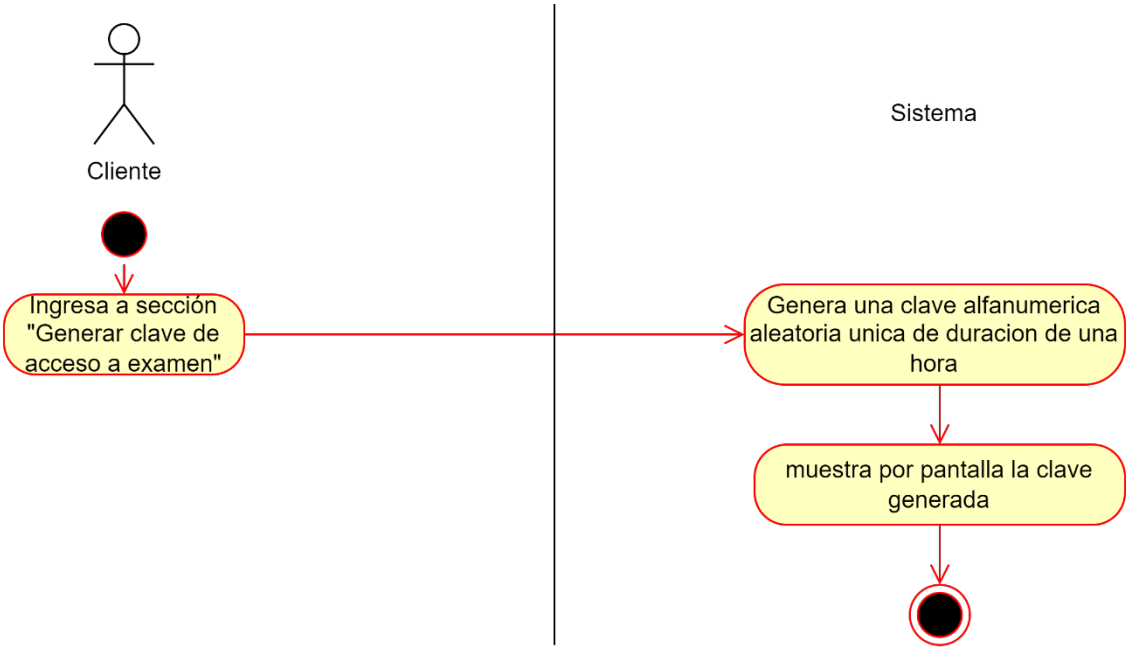
DACU2 – Usuario Registrándose



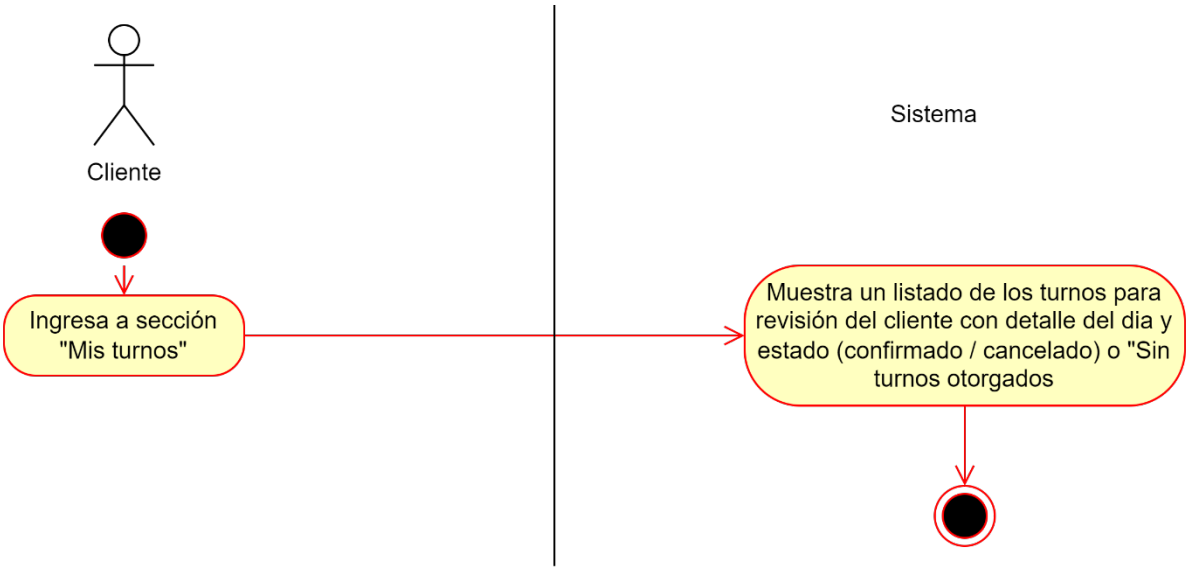
DACU3 – Cliente realizando Examen



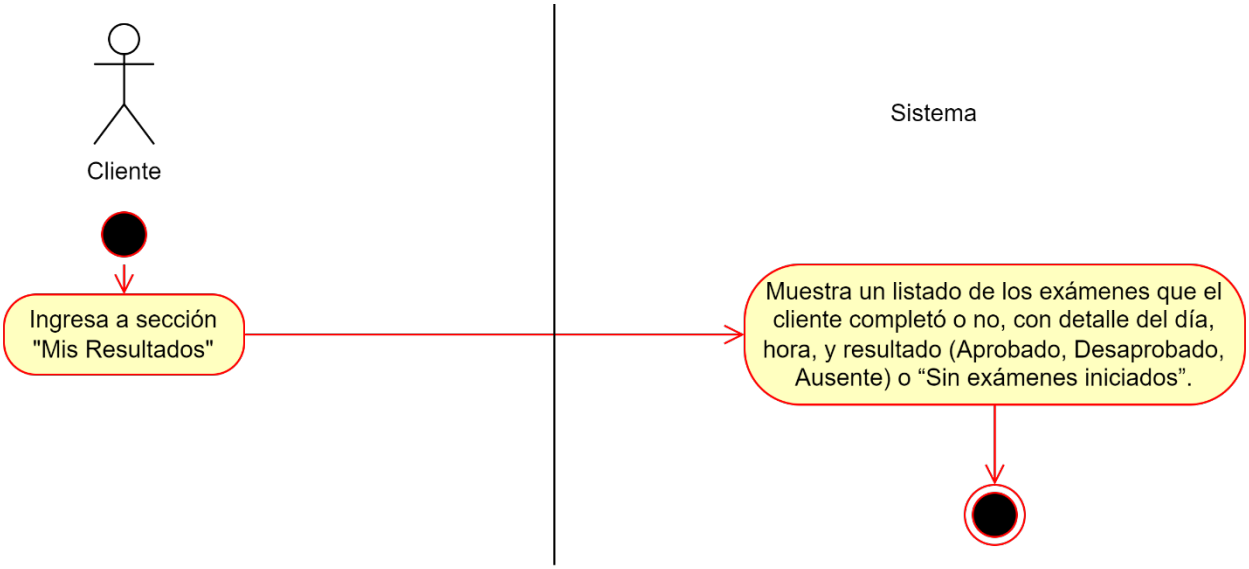
DACU4 – Cliente generando clave



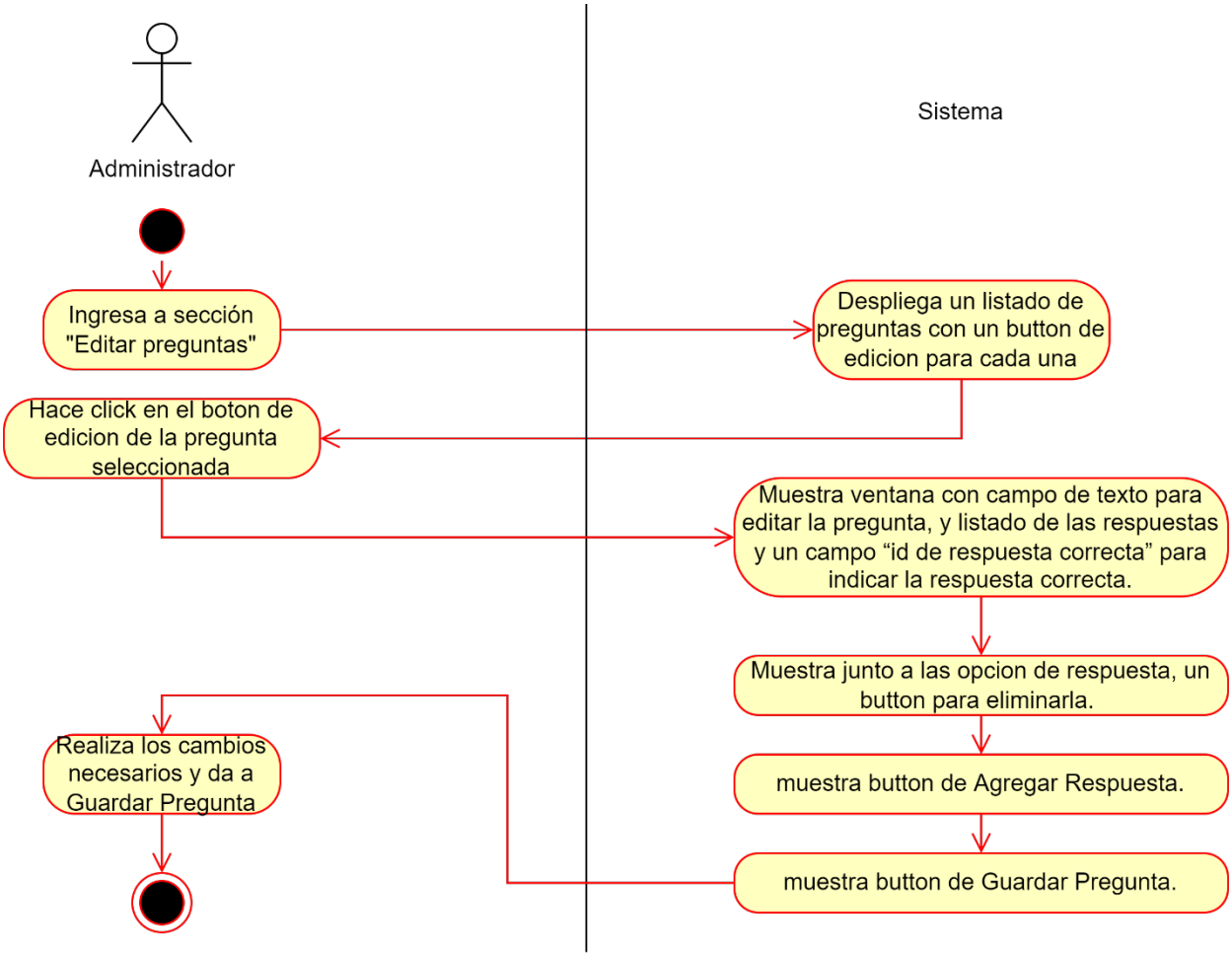
DACU5 – Cliente consultando turno



DACU6 – Cliente consultando resultados



DACU7 – Admin editando preguntas



DACU8 – Admin consultando estadísticas

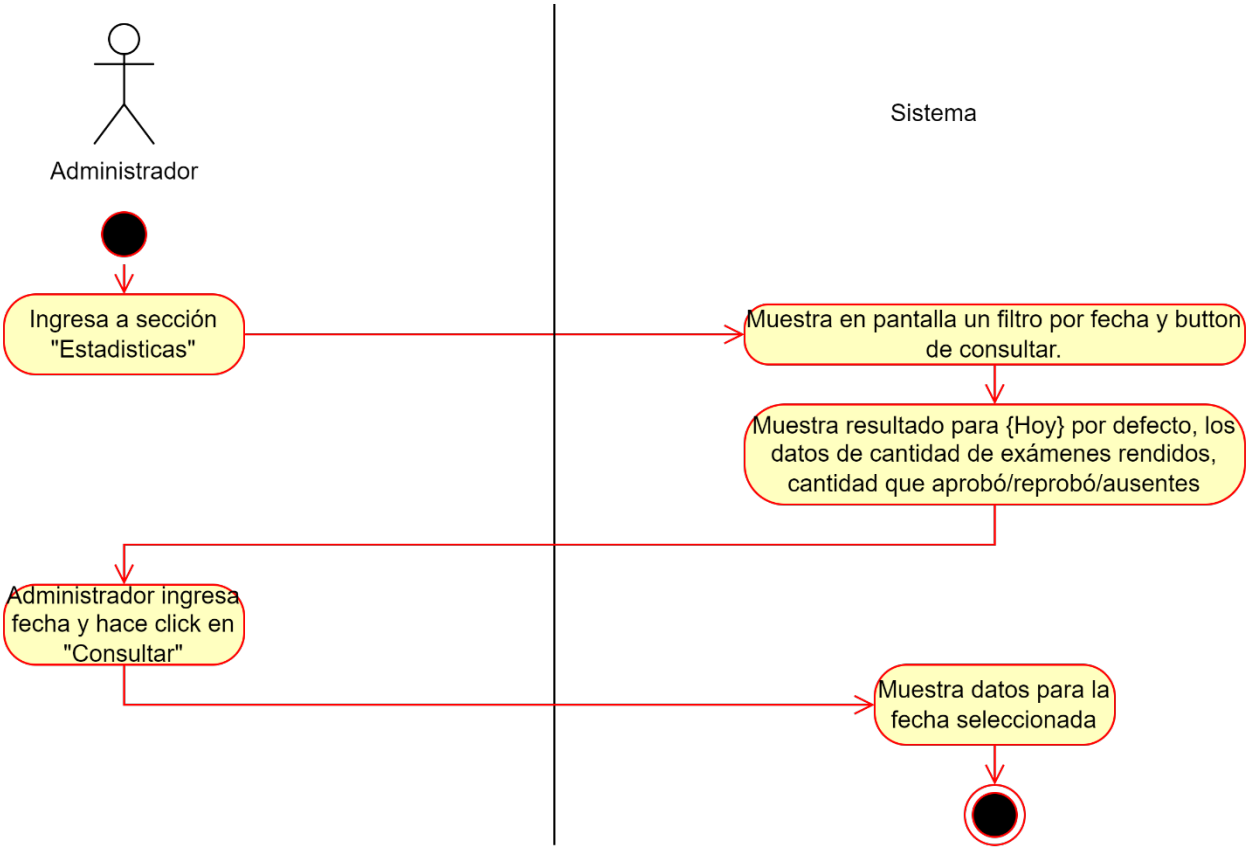
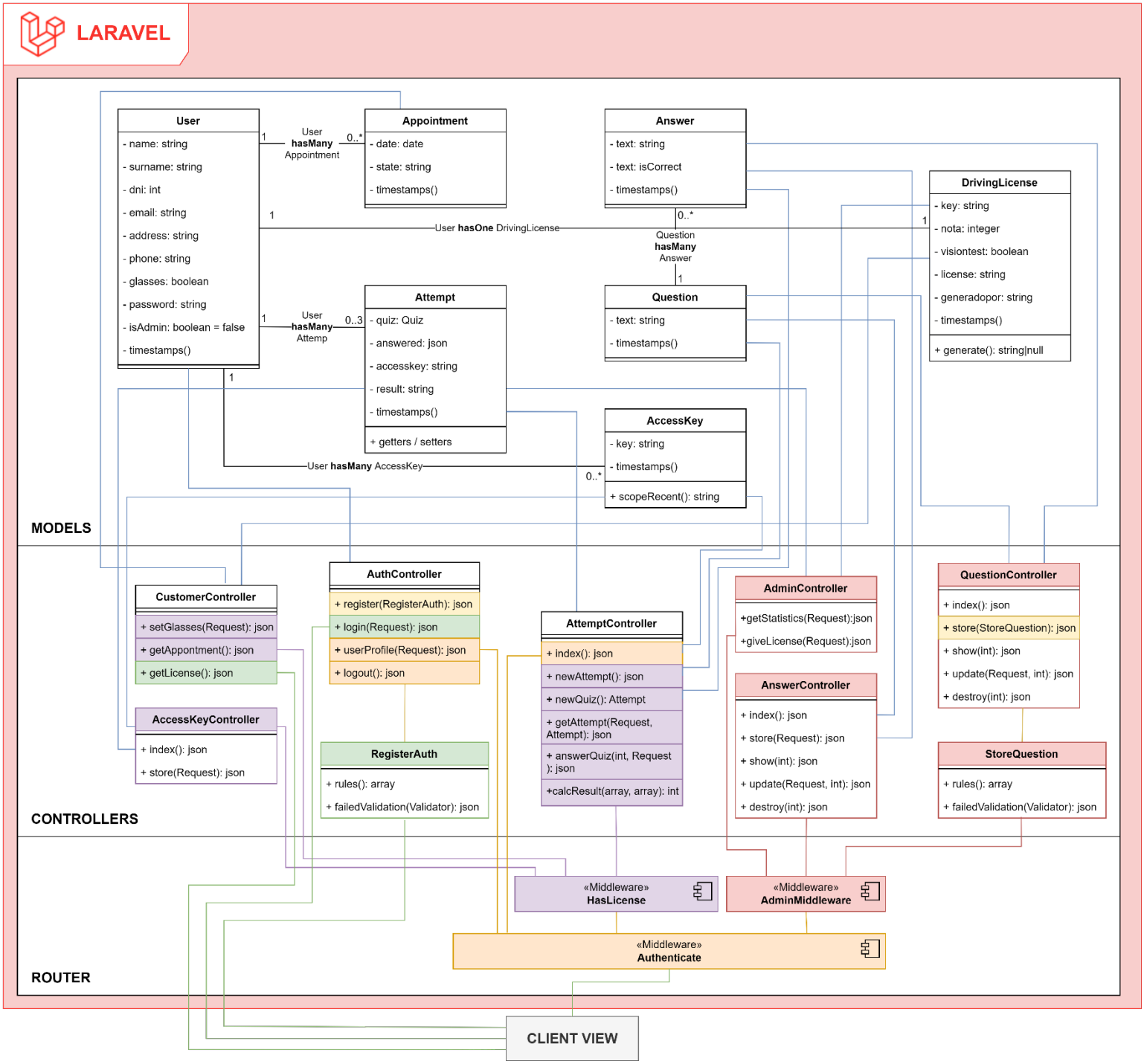
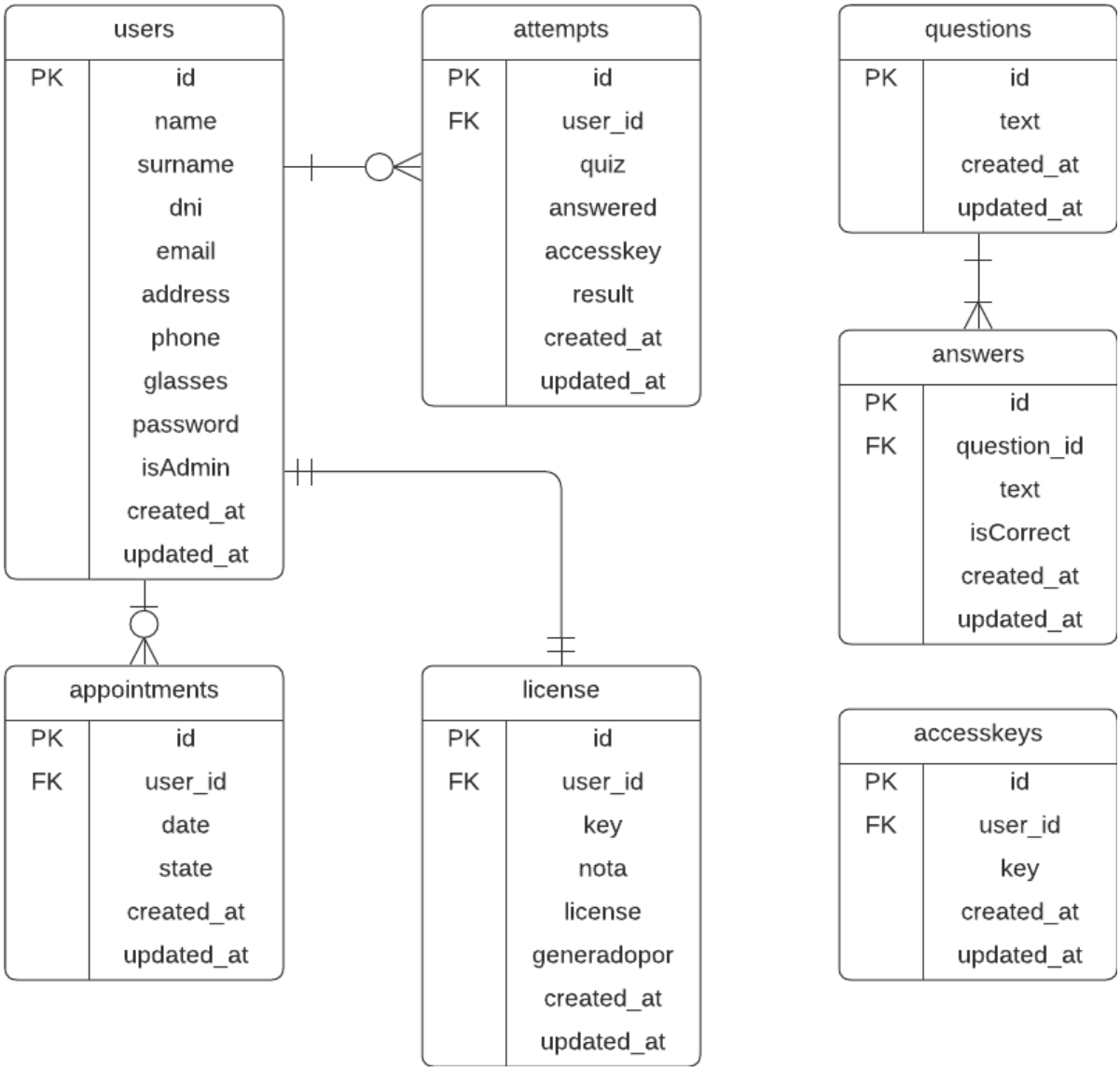


Diagrama de Clases

Diagrama MVC en el marco de Laravel (Hacer Zoom)



DER del Modelo de datos



Notas:

- Tanto quiz como answered, los valores se almacenarán en formato json.
- El accesskey se almacena por completo en la entidad attempts, independientemente de la entidad accesskey. Esto es para resguardar, aunque genere redundancia.

Detalle de tecnologías a utilizar

Desarrollo - PHP – Laravel

En base al trabajo realizado sobre comparación de frameworks, la libertad de selección de lenguaje y mi preferencia por el lenguaje PHP, para el desarrollo del sistema, se implementa el framework Laravel 9.
Laravel aporta las herramientas necesarias para la implementación de un sistema de usuarios, autenticación, estructura MVC y ORM.
El Backend será desarrollado completamente en el framework seleccionado siguiendo la arquitectura MVC.

La aplicación seguirá el modelo API Rest para poder ser consumida por cualquier cliente compatible con peticiones HTTP.

Persistencia – MySQL

MySQL es sistema de gestión de bases de datos relacional con licencia pública general, soportada por Laravel, y con una sintaxis simple.

Instalación

Requerimientos

El proyecto está desarrollado en el marco de Laravel 9.19 por lo tanto se requiere:

- ✓ PHP 8.0 o superior
- ✓ MySQL 5.7 o superior
- ✓ Composer
- ✓ Git (recomendado)

Repositorio: https://github.com/facundopalermo/tp-tap-v1

Situados en la carpeta tp-tap-v1, correr el instalador de composer para instalar las dependencias que se encuentran en el archivo composer.json

```
$ composer install
```

Correr el servidor que tenga disponible o utilizar el que provee Laravel con el comando

```
$ php artisan serve
```

Acto seguido, se debe configurar el archivo .ENV que contiene los parámetros de configuración de Laravel. En principio con configurar los datos de base de datos es suficiente para el testeo y funcionamiento del proyecto en un entorno localhost.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tap
DB_USERNAME=caishi
DB_PASSWORD=1234
```

Funcionalidades y explicación

A fin de explicar el funcionamiento y como se proveen los requerimientos funcionales requeridos, se seguirá una simple lógica de uso, donde primero el administrador configura las preguntas y luego el usuario común (customer) realiza los pasos para rendir el examen.

Registro y Autenticación

Los usuarios que desean hacer un examen deben primero registrarse para poder consumir los recursos. Esto se debe a que existe un middleware que solo permite el acceso a ciertos recursos si se está autenticado.

```
POST dominio/api/register
Body: JSON
{
    "name": "Juan",
    "email": "juan@mailfalso.com",
    "password": "1234",
    "password_confirmation": "1234",
    "surname": "Falso",
    "dni": 12345678,
    "address": "calle falsa 123",
    "phone": "+54911123456789"
}
```

Todos los campos son obligatorios según están definidos en el validador RegisterAuth.php

```
    public function rules()
    {
        return [
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'password' => 'required|confirmed',
            'surname' => 'required',
            'dni' => 'required|numeric|unique:users',
            'address' => 'required',
            'phone' => 'required'
        ];
    }
```

En caso de error, el endpoint lo retornara con el código HTTP correspondiente y un mensaje descriptivo, por ejemplo, que el DNI ya está en uso. Si el usuario se registra con éxito, el sistema retorna los datos del usuario creado.

A continuación, el usuario puede iniciar sesión con su email y contraseña.

```
POST dominio/api/login
Body: JSON
{
    "email": " juan@mailfalso.com",
    "password": "1234"
}
```

Si las credenciales son válidas, el sistema crea una cookie de sesión y retornará un token. Caso contrario informará que el usuario o la contraseña no son válidos.

```
{
    "token": "4|yWGfjY5Qz7roZTY9k8jXnN4DbkdMwlg3f11P98ps"
}
```

Como se indicó anteriormente, existe un middleware. Este es una implementación de Sanctum, el más simple de las herramientas de autenticación de Laravel que desde la version 9 está incluido. Además, hay implementado otro middleware que verifica si el usuario posee permisos de admin. (routes/api.php)

```
Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);

Route::middleware(['auth:sanctum'])->group(function () {

    /* USUARIO */
    Route::get('user-profile', [AuthController::class, 'userProfile']);
    Route::post('logout', [AuthController::class, 'logout']);

    Route::get('customers/attempts', [AttemptController::class, 'index']);
    Route::get('customers/license', [CustomerController::class, 'getLicense']);

    Route::group(['middleware' => 'haslicense'], function () {
        Route::post('customers/glasses', [CustomerController::class, 'setGlasses']);
        Route::get('customers/appointments', [CustomerController::class, 'getAppointment']);
        Route::get('customers/accesskey', [AccessKeyController::class, 'index']);
        Route::post('customers/accesskey', [AccessKeyController::class, 'store']);
        Route::post('customers/attempts', [AttemptController::class, 'newAttempt']);
        Route::get('customers/attempts/{attempt}', [AttemptController::class, 'getAttempt']);
        Route::post('customers/attempts/{id}', [AttemptController::class, 'answerQuiz']);
    });

    /* ADMIN */
    Route::group(['middleware' => 'admin'], function () {
        Route::get('statistics', [AdminController::class, 'getStatistics']);
        Route::post('license', [AdminController::class, 'giveLicense']);

        Route::resource('questions', QuestionController::class)
                ->only(['index', 'show', 'store', 'update', 'destroy']);

        Route::resource('answers', AnswerController::class)
                ->only(['index', 'show', 'store', 'update', 'destroy']);
    });

});
```

Adicionalmente, un usuario puede consultar su perfil de usuario haciendo

```
GET dominio/api/user-profile
```

Y finalizar la sesión haciendo

```
POST dominio/api/logout
```

Cabe resaltar que estas peticiones deber ser acompañadas del token si no se utiliza la cookie de sesión.

Como se indicó, anteriormente, la forma de controlar la Autorización es mediante un middleware, entonces los usuarios admin contienen en su clase el atributo booleano isAdmin, el cual mediante el middleware AdminMiddleware se verifica que el usuario este autenticado y tenga el atributo en true. Caso contrario, no permite el acceso.

```
    public function handle(Request $request, Closure $next) {
        if (auth()->check() && auth()->user()->isAdmin) {
            return $next($request);
        }

        return response(["message"=>"Usuario sin permisos"], Response::HTTP_FORBIDDEN);
    }
```

Administración de Preguntas

Solo los administradores pueden ver, crear, modificar o eliminar preguntas

Crear

Se puede crear una pregunta con sus múltiples respuestas de la siguiente manera

```
POST dominio/api/questions
Body: JSON
{
    "text": "Pregunta?",
    "answers":[
        {
            "text": "Respuesta 1",
            "isCorrect": true
        },
        {
            "text": "Respuesta 2",
            "isCorrect": false
        }
    ]
}
```

Modificar

Para modificarla, basta con hacer put al recurso. Solo admite modificar el texto. Para cambiar las respuestas se debe hacer mediante el recurso answers.

```
PUT dominio/api/questions/{id}
Body: JSON
{
    "text": "Pregunta modificada?",
}
```

Eliminar

Y para eliminarla, mediante el verbo delete. Es importante señalar que la eliminación es por cascada, por lo tanto al eliminar una pregunta, se eliminan sus respuestas asociadas.

```
DELETE dominio/api/questions/{id}
```

Obtener

Para obtener todas las preguntas con sus respuestas, o tan solo una pregunta en particular, basta obtenerlas mediante GET

```
GET dominio/api/questions
GET dominio/api/questions/{id}
```

Administración de Respuestas

Solo los administradores pueden ver, crear, modificar o eliminar respuestas

Crear

Se puede crear una respuesta nueva para una pregunta, indicando el id de la pregunta

```
POST dominio/api/answers
Body: JSON
{
    "text": "Respuesta nueva",
    "question_id": 1,
    "isCorrect": false
}
```

Modificar

Para modificarla, basta con hacer put al recurso. Solo admite modificar el texto. Para cambiar las respuestas se debe hacer mediante el recurso answers.

```
PUT dominio/api/answers/{id}
Body: JSON
{
    "text": "Respuesta modificada?",
}
```

Eliminar

Y para eliminarla, mediante el verbo delete.

```
DELETE dominio/api/answers/{id}
```

Obtener

Para obtener todas las respuestas, o tan solo una respuesta en particular, basta obtenerlas mediante GET

```
GET dominio/api/answers
GET dominio/api/answers/{id}
```

Obtener estadísticas

Los administradores pueden obtener las estadísticas de cantidad de exámenes realizados en el día, los aprobados, desaprobados y ausentes mediante el recurso statistics.

En el body se puede enviar una fecha (Y-m-d). De no hacerlo, por default devolverá los datos del día de la fecha.

```
GET dominio/api/statistics
Body: JSON (optional)
{
    "date": "2022-11-20"
}
```

Uso de anteojos.

Los clientes (customer) antes de iniciar un cuestionario deben indicar si usan o no anteojos según AttempController

```
if($user->glasses == null) {
    return response()->json(['message' => 'post glasses es requerido'], Response::HTTP_NOT_ACCEPTABLE);
}
```

Para ello, se debe hacer

```
POST dominio/api/customers/glasses
Body: JSON
{
    "glasses" : true
}
```

Si se indica true, el sistema genera un turno con fecha aleatoria entre el día de la fecha y un año (CustomerController.setGlasses)

```
if($user->glasses == true) {
    if(!$appointment = Appointment::where('user_id', $user->id)->where('state', 'valid')->where('date',
'>', date('Y-m-d'))->first()){
        $date = fake()->dateTimeBetween('now', '+1 years')->format('Y-m-d');
        $appointment = Appointment::create([
            'user_id' => $user->id,
            'date' => $date,
            'state' => 'valid'
        ]);
    }
}
```

Consultar turnos

Adicionalmente, el cliente puede consultar el historial de sus turnos haciendo

```
GET dominio/api/customers/appointments
```

Exámenes

El objetivo principal del sistema es la generación de exámenes de preguntas y respuestas para que los clientes las respondan y obtengan un resultado.

Generar nuevo intento

Al generar un intento nuevo, se genera un examen. Para ello se debe

```
POST dominio/api/customers/attempts
```

El Sistema hará algunas validaciones y resolverá según el resultado.

1) Verificar que se haya contestado si usa lentes.

2) Si existe un intento sin utilizar, devuelve el id. Se entiende un intento si usar cuyos atributos answered, accesskey y result son NULL.

3) Si no existe intento sin utilizar, y no se superan los 3 intentos registrados, crea un intento y lo retorna.

4) Si se superan los 3 intentos, se informa.

El examen que se genera es único para cada intento y cada intento es único para cada cliente, obteniendo en forma aleatoria 10 preguntas de la base de datos, cuyas respuestas también están en orden aleatorio (AttemptController.newQuiz).

```
$questions = Question::with(['answers'=> function ($q) {
    $q->inRandomOrder();
}])->inRandomOrder()->limit(10)->get();

$attempt = Attempt::create([
    'user_id' => $user->id,
    'quiz' => $questions
]);
```

Consultar intentos

Para consultar los datos de los intentos generados disponibles, respondidos y ausentes, de haberlos, hacer

```
GET dominio/api/customers/attempts
```

Retorna un json de intentos con el id, el resultado (null, Aprobado, Reprobado, Ausente) y la fecha de creación.

Acceder al examen

Acceder al examen significa realizar un intento. Para ello se debe

```
GET dominio/api/customers/attempts/{id}
Body: JSON
{
    "accesskey": "MChb45kBCclpYC6s6kpN"
}
```

Como se puede observar, requiere un clave de acceso la cual es única y una vez usada no se puede volver a utilizar (ver sección Obtener clave de acceso).

El Sistema hará algunas validaciones y resolverá según el resultado.

1) Que el intento y examen correspondan al usuario.

2) Que el examen ya no haya sido accedido. Este GET solo se puede hacer una vez (correctamente). Hecho, se registra como leído con la clave de acceso.

3) Que el accesskey este vigente y le pertenezca al usuario. Para ello utiliza una función que crea un query where para validar que sea reciente (menor a una hora)

```
$accesskey = AccessKey::recent()->where('key', $request->accesskey)->where('user_id', $user->id)->first();
```

Donde recent() es la función scopeRecent() de la clase modelo

```
    public function scopeRecent($query) {
        return $query->whereDate('created_at' , '=', Carbon::today())
                     ->whereTime('created_at' , '>',Carbon::now()->subHours(1));
    }
```

4) Verificado 3), además verifica que la accesskey no haya sido utilizada en otro intento.

Si pasa todos los filtros, se registra la accesskey en el intento, y retorna un json con el id del intento y la guía de 10 preguntas y respuestas:

```
{
    "result": {
        "attempt:": 8,
        "quiz": [
            {
                "id": 14,
                "text": "Todo usuario de la vía pública debe, como premisa básica",
                "answers": [
                    {
                        "id": 40,
                        "text": "Concurrir a cursos de actualización en temática ..."
                    },
                    {
                        "id": 42,
                        "text": "Asumir la obligación de no generar peligro innecesario."
                    },
                    {
                        "id": 41,
                        "text": "Acreditar experiencia de manejo en vehículos, que ..."
                    }
                ]
            },
            {...} x 10
        ]
    }
}
```

Responder examen

Una vez obtenido el examen siguiendo el procedimiento del punto anterior, se debe responder dentro de una hora de la generación de este.

Para ello se debe enviar las respuestas de la siguiente manera:

```
POST dominio/api/customers/attempts/{id}
Body: JSON
{
    "accesskey": "MChb45kBCclpYC6s6kpN",
    "responses": [
        {
            "question": 3,
            "answer": 8
        },
        {
            "question": 9,
            "answer": 27
        },
        { ... } x 10
    ]
}
```

Hay algunas validaciones a tener en cuenta, en primer lugar, el request

```
$request->validate([
    'accesskey' => 'required|string|min:20|max:20',
    'responses' => 'array|required|size:10',
    'responses.*.question' => 'required|integer',
    'responses.*.answer' => 'required|integer',
]);
```

Como se puede ver, todos los campos son obligatorios y debe haber 10 preguntas con su respuesta, la cual es el id de la respuesta correcta (isCorrect=true).

Se valida que el cuestionario sea del usuario y que la clave de acceso sea la correcta.

También el sistema valida que el cuestionario exista, y que además no haya sido evaluado anteriormente.

```
if($attempt->answered != null || $attempt->result != null) {
    return response()->json(['message' => 'Cuestionario ya evaluado'], Response::HTTP_BAD_REQUEST);
}
```

Resultado del examen

Superados los filtros de validación, se calcula los resultados y según sea la nota, se registra fue Aprobado o Reprobado. En el segundo escenario, se intentará generar un nuevo examen, conforme a las reglas que para ello se requiere.

```
$nota = self::calcResult((array) json_decode($attempt->quiz), $request->responses);

/* prepara response */
$result = array();
$result['hits'] = $nota;

/* evalua la nota */
if($nota > 7){
    $result['result'] = 'Aprobado';
}else{
    $result['result'] = 'Reprobado';

    if ($newAttempt = self::newQuiz()) {
        $result['new_attempt'] = $newAttempt->id;
    }else{
        $result['new_attempt'] = 'Cantidad de intentos superados';
    }
}
```

```
public function calcResult(array $quiz, array $responses): int {
    $points = 0;
    foreach ($quiz as $question) {
        foreach ($responses as $response) {
            if($question->id == $response['question']){
                $points += (Answer::findorfail($response['answer']))->isCorrect? 1 : 0;
                break;
            }
        }
    }
    return $points;
}
```

Clave de Acceso

Cada examen requiere una clave única de acceso. La misma tiene una duración de una hora. Solo se puede utilizar en un intento, y se la requiere enviar tanto para acceder al examen como para enviar las respuestas. Están asociadas a un usuario.

Para generar una clave hacer POST al recurso. Si hay clave vigente no utilizada, retorna la clave.

```
POST dominio/api/customers/accesskey
```

Para consultar si hay una clave vigente GET

```
GET dominio/api/customers/accesskey
```

Generación de Licencia

Cuando el cliente contesta por primera vez si utiliza anteojos, el sistema asocia un modelo de DrivingLicense al usuario. Este modelo es el que, a medida que avance, se registrara la licencia y los datos asociados a ella.

Primero, si contesta que no debe usar anteojos, el sistema guarda true en la propiedad visiontest (en términos del programa, significa si pasa el test de visión), caso contrario queda en null.

Luego, si aprueba el examen se completan los atributos key = accesskey, la nota, y no debe usar lentes (visiontest = true) y se genera una licencia a partir de estos datos utilizando md5, lo que garantiza que sea única.

```
$this->generadapor = auth()->user()->id;
$str = $this->id . $this->key . $this->nota . $this->created_at;
$this->license = hash('md5', $str);
$this->save();
```

En el caso de que se trate de un cliente que deba usar lentes y concurrir al examen ocular, un administrador establecerá en true el visiontest, y si aprobó el examen, se emite la licencia.

Para ello, se debe consumir el siguiente recurso

```
POST dominio/api/license
Body: JSON
{
    "customer_id": {id}
}
```

Finalmente, un usuario puede consultar su licencia haciendo

```
GET dominio/api/customers/license
```

Middleware “hasLicense”

Este middleware restringe el acceso a los recursos clave de acceso, intentos, examen, etc. si es el cliente ya tiene una licencia otorgada. Un cliente puede tener solo una licencia de conducir. El middleware podría ser utilizado para permitir acceso a esos recursos si la licencia se venciera, extraviara, etc. (Situaciones no contempladas en este proyecto.)

Tareas programadas

Laravel permite crear tareas programadas para que, al ejecutar ciertos comandos, se realicen.

La configuración de cron job depende del sistema operativo, sin embargo, a fines de este proyecto y en el ámbito de desarrollo, basta con correr en una terminal el siguiente comando para iniciar la cola de tareas. Las tareas se configuran en app/Console/Kernel.php

```
$ php artisan schedule:work
```

En caso de que se quiera correr todas las tareas sin esperar, hacer

```
$ php artisan schedule:run
```

Para ver la lista de tareas programadas, hacer

```
$ php artisan schedule:list
```

Limpieza de resultados (daily)

La clase encargada de la eliminar los resultados de hace mas de una semana de antigüedad se encuentra en app/Console/Commands/ClearResult.php.

La función simplemente, elimina aquellos intentos que hayan sido generados siete días antes del día de la fecha, y guarda la acción en ClearResults.log

```
    public function handle() {
        Attempt::where('created_at', '<', Carbon::now()->subDays(7))->delete();
        $log = '[' . date("Y-m-d H:i:s") . ']: ' . "Registro de resultados borrado. ";
        Storage::append("ClearResults.log", $log);
    }
```

Clientes Ausentes (everyMinute)

La clase encargada de la marcar Ausentes a aquellos intentos que no fueron respondidos en una hora se encuentra en app/Console/Commands/QuizTimeOut.php.

La función es simplemente, actualiza aquellos intentos que no fueron respondidos en una hora, marcando Ausente el result, y guarda la acción en QuizTimeOut.log

```
    public function handle() {
        Attempt::where('created_at', '<', Carbon::now()->subMinutes(60))->where('answered', NULL)
                ->whereNotNull('accesskey')->update(['result' => 'Ausente']);
        $log = '[' . date("Y-m-d H:i:s") . ']: ' . "Intentos sin intentar, como ausentes. ";
        Storage::append("QuizTimeOut.log", $log);
    }
```

Testing

Para los test se ha utilizado la herramienta PHPUnit incluida en el framework de Laravel.

Los mismos se encuentran en el directorio tests separados en dos Modos:

1. Feature:
    a. AccessKeyTest: Son dos (2) tests para evaluar la creación y obtención de una accesskey
    b. AuthTest: cinco (5) tests para evaluar el login, obtención del perfil y los permisos de administrador sobre un recurso.
2. Unit:
    a. CalcResultTest: única prueba sencilla del método calcResult del controlador de Intentos (AttemptController::calcResult())

Para correrlos basta con hacer

```
$ php artisan test
```

o, si se desea un modo mas simple y directo

```
$ ./vendor/bin/phpunit
```

Seeders

Opcionalmente, se puede correr el DatabaseSeeder que genera dos usuarios (un admin) y 15 preguntas con sus posibles respuestas para el examen.

Para usarlo, hay dos formas:
Al hacer la migración:

```
php artisan migrate:fresh –seed
```

o, luego de la migración

```
php artisan db:seed
```

Registro de cambios

#	fecha	Detalle
1	02-10-2022	Emisión. Primer documento. Entrega primera etapa
2	22-10-2022	La autenticación del usuario ahora es con email y contraseña. Antes era con usuario/contraseña. Se realiza el cambio ya que es requisito ingresar un mail para registrarse y este es único, no hay necesidad de un username extra.
	22-10-2022	Para el rol de admin, se cambio patrón de roles por middleware que verifica si el usuario isAdmin, ya que implementar un sistema de roles y permisos para esta aplicación, llevaría mucho trabajo y estudio de funcionamiento (Spatie por ejemplo)
3	24-10-2022	Se quitó respuesta indicada en CU4 p2 que el sistema muestra hora de caducidad de la clave.
4	20-11-2022	Agregado cron job al diagrama de actividades DACU3, para marcado Ausente en exámenes no finalizados en una hora.
5	20-11-2022	Actualizado el diagrama de clases en el marco de Laravel, utilizando MVC.
6	20-11-2022	Actualizado el DER del modelo de datos
7	20-11-2022	Actualizado el desarrollo, ya que no se implementa en este proyecto el frontend, solo se desarrolla el backend mediante API Rest.
8	21-11-2022	Generada toda la documentación de uso
9	23-11-2022	Agregada documentación sobre licencia