Lenin Córdova

Instituto Tecnológico Superior Universitario del

Azuay Desarrollo de aplicaciones Web

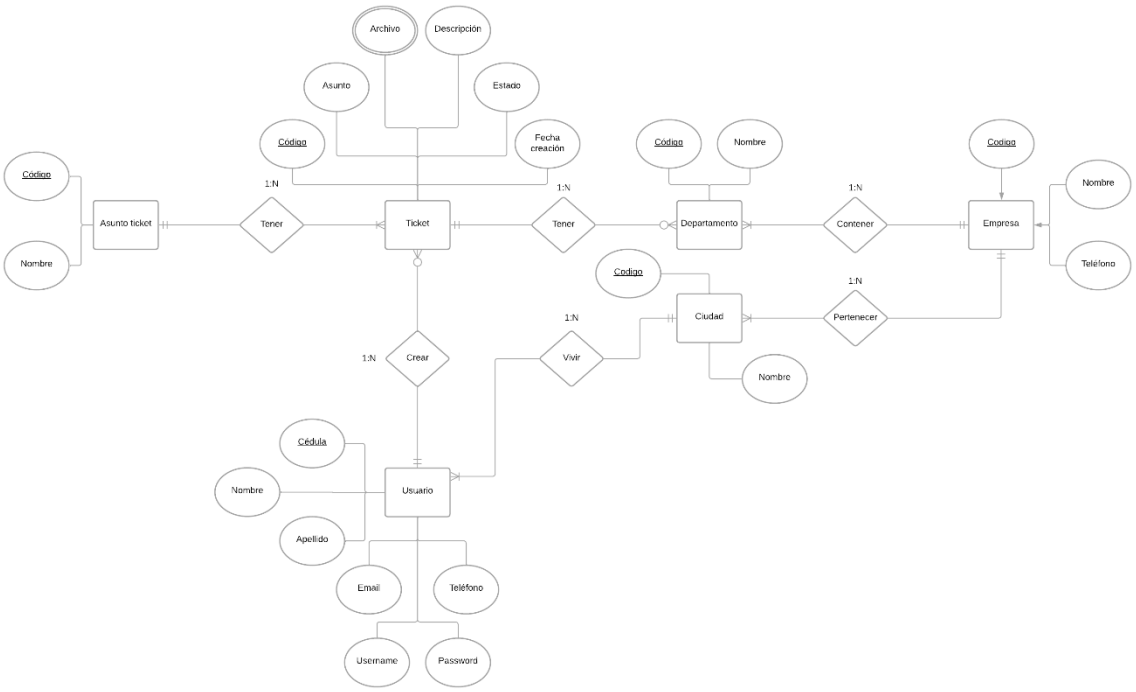
Ing. Carmen Tacuri, Mgtr.

Marzo 10, 2024

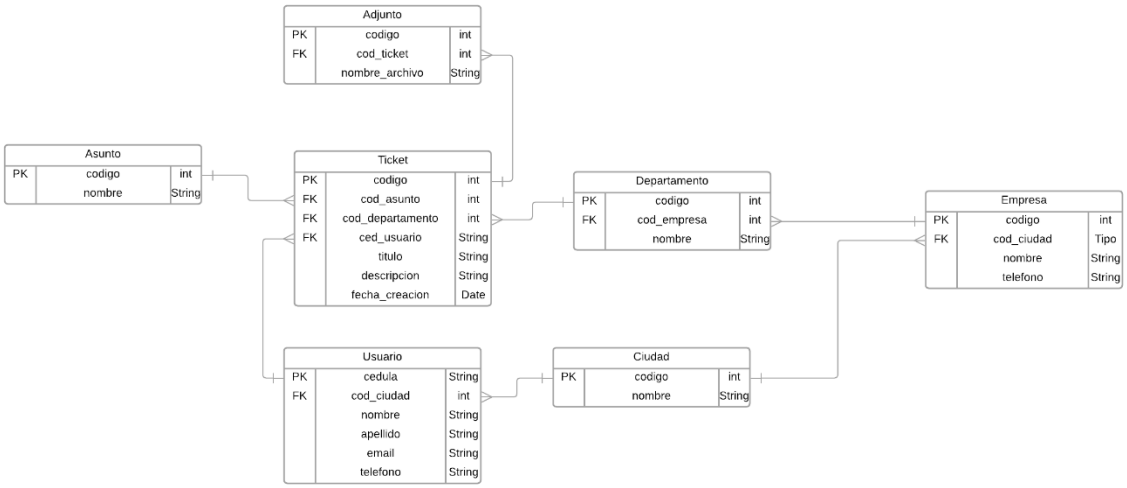
Se busca desarrollar una Aplicación Web capaz de permitir la creación de tickets y varias funciones más.

Realizando un análisis profundo de la aplicación se puede obtener los siguientes resultados:

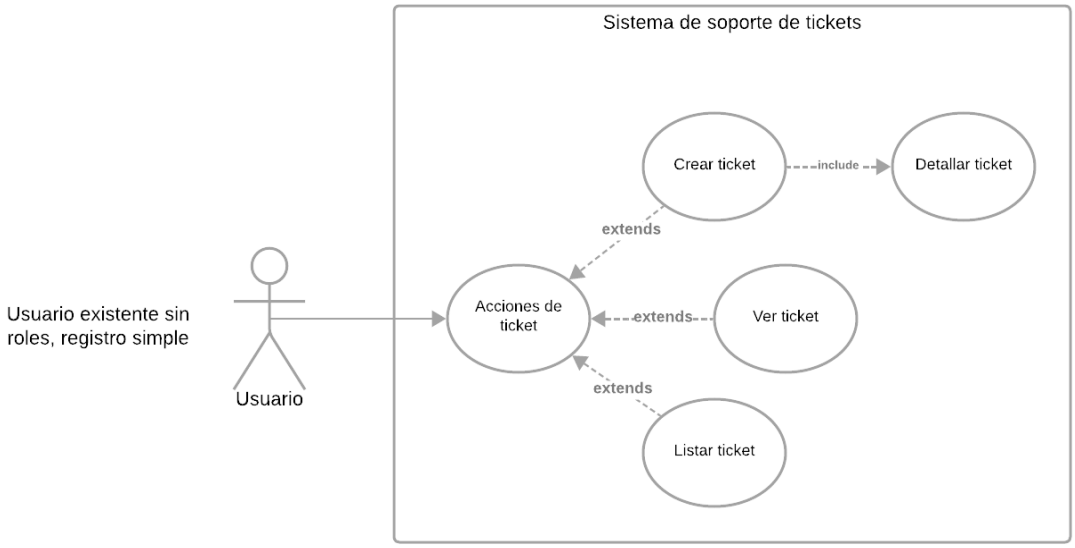
Modelo Entidad – Relación:



Modelo Relacional:



Caso de uso:



Por lo que, el sistema debe cumplir una función principal, poder crear, ver ticket y listar los tickets.

Aclaraciones:

1.- Hay cambios con respecto a los requisitos pedidos principalmente:

Código fuente desarrollado en Spring utilizando la base de datos embebida h2.

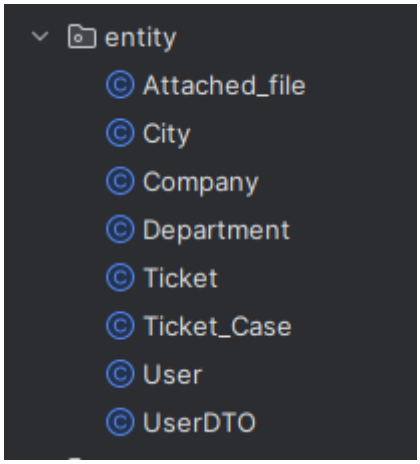
No se llevó a cabo con la base de datos H2, porque la creación en esta base de datos con las entidades tenía un error, no se creaba correctamente. Así que se optó por cambiar a MySQL.

2.- No se desarrolló con WebSockets para permitir las respuestas a los tickets creados en tiempo real por falta de tiempo.

Informe del desarrollo del aplicativo:

La aplicación de Spring Boot cuenta con varias dependencias necesarias para el desarrollo, entre las más destacadas son: MySQL, Lombok, Spring Web, Dev Tools, ThymeLeaf y Web Sockets.

Se llevó a cabo el desarrollo de la aplicación con el patrón MVC a partir del modelo relacional mencionado anteriormente, así que, se desarrollaron varias entidades:



Estas entidades están definidas de la siguiente manera:

1.- Archivo adjunto

```java
@Entity
@Data
public class Attached_file implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long codeFile;

    private String fileName;

    @ManyToOne
    @JoinColumn(name = "code_ticket", referencedColumnName = "codeTicket")
    private Ticket codeTicket;
}
```

2.- Ciudad

```java
@Entity
@Data
public class City implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long codeCity;

    private String name;
}
```

3.- Empresa

```java
@Entity
@Data
public class Company implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long codeCompany;

    private String name;

    private String phone;

    @ManyToOne
    @JoinColumn(name = "code_city", referencedColumnName = "codeCity")
    private City codeCity;
}
```

4.-Departamento

```java
@Entity
@Data
public class Department implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long codeDepartment;

    private String name;

    @ManyToOne
    @JoinColumn( referencedColumnName = "codeCompany", name = "code_company")
    private Company codeCompany;
}
```

5.- Ticket

```java
@Entity
@Data
public class Ticket implements Serializable {

    @Id
    private Long codeTicket;

    private String title;

    private String description;

    private Boolean state;

    @Temporal(TemporalType.DATE)
    private Date createAt;

    @ManyToOne
    @JoinColumn(name = "code_case", referencedColumnName = "codeCase")
    private Ticket_Case ticketCase;

    @ManyToOne
    @JoinColumn(name = "code_department", referencedColumnName = "codeDepartment")
    private Department codeDepartment;

    @ManyToOne
    @JoinColumn(name = "id_card", referencedColumnName = "idCard")
    private User userIdCard;

    @PrePersist
    public void prePersist() {
        state = Boolean.TRUE;
        createAt = new Date();
        codeTicket = generarNumeroAleatorioDe5Cifras();
    }

    1 usage
    private Long generarNumeroAleatorioDe5Cifras() {
        Random random = new Random();
        return (long) (10000 + random.nextInt( bound: 90000));
    }

}
```

Esta clase contiene 2 métodos adicionales, realizados para generar un valor aleatorio de 5 cifras y predefinir algunos valores con @PrePersist.

6.- Caso de ticket

```java
@Entity
@Data
public class Ticket_Case implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long codeCase;

    private String name;
}
```

7.- Usuario

```java
@Entity
@Data
public class User implements Serializable {

    @Id
    private String idCard;

    private String name;

    private String lastName;

    private String email;

    private String phone;

    private String password;

    @ManyToOne
    @JoinColumn(name = "codeCity")
    private City codeCity;
}
```

8.- UserDTO

```java
@Data
public class UserDTO implements Serializable {
    private String idcard;
    private String password;
}
```

Esta clase adicional fue realizada para ser utilizada dentro un pequeño formulario semejante a un inicio de sesión.

Dao y Service (Incluye implementaciones de las interfaces)

Cada Dao y Service en su clase interface contiene los 4 métodos básicos de un CRUD.

```
4 usages  1 implementation
public interface ICase {

    1 implementation
    public List<Ticket_Case> findAll();

    1 implementation
    public void save(Ticket_Case ticketCase_ticket);

    1 implementation
    public Ticket_Case findOne(Long id);

    1 implementation
    public void delete(Long id);
}
```

Y en su implementación en una clase es de la siguiente manera, manteniendo toda una estructura similar a la mostrada a continuación:

```
@Repository
public class ICaseImpl implements ICase {

    5 usages
    @PersistenceContext
    private EntityManager em;

    /unchecked/
    @Override
    public List<Ticket_Case> findAll() { return em.createQuery( qlString: "from Ticket_Case").getResultList(); }

    @Override
    public void save(Ticket_Case ticketCase_ticket) {
        if (ticketCase_ticket.getCodeCase() != null && ticketCase_ticket.getCodeCase() > 0) {
            em.merge(ticketCase_ticket);
        } else {
            em.persist(ticketCase_ticket);
        }
    }

    @Override
    public Ticket_Case findOne(Long id) { return em.find(Ticket_Case.class, id); }

    @Override
    public void delete(Long id) { em.remove(findOne(id)); }
}
```

En donde, a partir de la clase “EntityManager” podemos encargarnos de gestionar las entidades y sus relaciones con la base de datos. Es una herramienta fundamental para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en entidades persistentes.

Mientras que, por otro lado, en la implementación de Service, es de la siguiente manera:

```java
@Service
public class ICaseServiceImpl implements ICaseService {

    @Autowired
    private ICase caseDao;

    @Override
    @Transactional(readOnly = true)
    public List<Ticket_Case> findAll() { return caseDao.findAll(); }

    @Override
    @Transactional
    public void save(Ticket_Case ticketCase_ticket) { caseDao.save(ticketCase_ticket); }

    @Override
    @Transactional(readOnly = true)
    public Ticket_Case findOne(Long id) { return caseDao.findOne(id); }

    @Override
    @Transactional
    public void delete(Long id) { caseDao.delete(id); }
}
```

Donde la principal característica es que se usa la Interface del Dao para permitir el acceso y controlar la comunicación de la aplicación la base de datos.

Controlador:

```
@Controller
@RequestMapping("/ticket")
public class TicketController {

    @Autowired
    private ITicketService ticketService;

    @Autowired
    private IAttachedFileService attachedFileService;

    @Autowired
    private IDepartmentService departmentService;

    @Autowired
    private IUserService userService;

    @Autowired
    private ICaseService ticketCaseService;

    @Value("${spring.config.location}")
    private String URL;

    @RequestMapping(value = {"/list-all-tickets", "", "/", "/index",
"/inicio"})
    public String getTickets(Map<String, Object> model, HttpSession
session) {
        String requestSession = (String)
session.getAttribute("idCard");

        if (requestSession == null || requestSession.isEmpty()) {
            return "redirect:/user/login";
        }

        model.put("title", "List All tickets");
        model.put("session", requestSession);
        model.put("tickets", ticketService.findAll());

        return "table/list-ticket";
    }

    @RequestMapping(value = "/save-ticket")
    public String createTicket(Map<String, Object> model, HttpSession
session) {

        Ticket ticket = new Ticket();
        Attached_file file = new Attached_file();
        String requestSession = (String)
session.getAttribute("idCard");

        model.put("title", "Open new ticket");
        model.put("ticket", ticket);
        model.put("files", file);
        model.put("user", userService.findOne(requestSession));
        model.put("departments", departmentService.findAll());
        model.put("ticketCases", ticketCaseService.findAll());

        return "form/form-ticket";
    }
```

```
    @PostMapping("/save-ticket")
    public String saveTicket(Ticket ticket, RedirectAttributes flash,
@RequestParam(value = "file", required = false) MultipartFile file,
HttpSession session) {

        if (!file.isEmpty()) {
            try {
                Path RutaArchivo = Paths.get(URL + "//" +
file.getOriginalFilename());
                Files.write(RutaArchivo, file.getBytes());
                flash.addFlashAttribute("info", "Se ha subido
correctamente la foto '" + file.getOriginalFilename() + "'");

            } catch (IOException e) {
                flash.addFlashAttribute("danger", "Error al guardar la
foto: " + e.getMessage());
                return "redirect:/ticket/list-all-tickets";
            }

        }

        User user = userService.findOne((String)
session.getAttribute("idCard"));
        ticket.setUserIdCard(user);
        System.out.println(ticket);

        Long idTicket = ticketService.save(ticket);
        ticket.setCodeTicket(idTicket);
        Attached_file newFile = new Attached_file();
        newFile.setCodeTicket(ticket);
        newFile.setFileName(file.getOriginalFilename());
        flash.addFlashAttribute("success", "Se ha guardado la foto");

        attachedFileService.save(newFile);

        return "redirect:/ticket/list-all-tickets";
    }

    @GetMapping("/ver-ticket/{id}")
    private String seeDetails(@PathVariable("id") Long id, Map<String,
Object> model, RedirectAttributes flash) {

        Ticket ticket = ticketService.findOne(id);
        Attached_file file = attachedFileService.findOneByTicket(id);

        if (ticket == null) {
            flash.addFlashAttribute("danger", "El ticket no existe en
la base de datos");
            return "redirect:/ticket/list-all-tickets";
        }

        model.put("title", "Ver ticket N°: " +
ticket.getCodeTicket());
        model.put("ticket", ticket);
        model.put("file", file);

        return "card/card-ticket";
    }
}
```

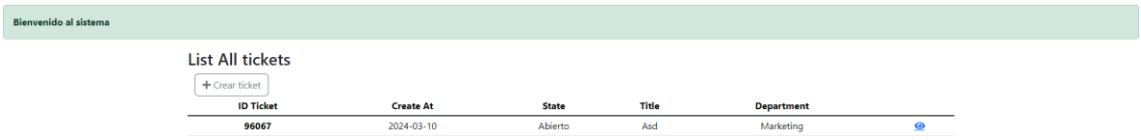
Para entender el código del controlador vamos a imaginar que este código es como un asistente virtual que ayuda a manejar los tickets de soporte técnico en una empresa. Cada método es como una tarea diferente que el asistente puede realizar.

1. getTickets(...):
    - Esta es la tarea principal del asistente, donde muestra todos los tickets abiertos en una lista.
    - Primero, verifica si el usuario ha iniciado sesión. Si no, lo redirige al inicio de sesión.
    - Luego, prepara los datos necesarios (título, lista de tickets, etc.) y los organiza en un modelo.
    - Finalmente, muestra la lista de tickets en una tabla.
2. createTicket(...):
    - Esta tarea es como cuando un usuario quiere abrir un nuevo ticket de soporte.
    - El asistente prepara un formulario en blanco para que el usuario llene los detalles del problema.
    - También obtiene información adicional, como los departamentos disponibles y los tipos de casos.
    - Todo esto se organiza en un modelo y se muestra en el formulario.
3. saveTicket(...):
    - Aquí es donde el asistente guarda el nuevo ticket que el usuario acaba de crear.
    - Primero, verifica si el usuario adjuntó algún archivo (como una captura de pantalla).
    - Si hay un archivo, lo guarda en el servidor y muestra un mensaje de confirmación.
    - Luego, toma los detalles del ticket y los asocia con el usuario que lo creó.
    - Guarda el ticket en la base de datos y también guarda el archivo adjunto si lo hay.
    - Finalmente, redirige al usuario a la lista de tickets nuevamente.
4. seeDetails(...):
    - Esta tarea es cuando un usuario quiere ver los detalles de un ticket específico.
    - El asistente busca el ticket en la base de datos usando su identificador.
    - Si no encuentra el ticket, muestra un mensaje de error y redirige al usuario a la lista de tickets.
    - Si encuentra el ticket, obtiene los detalles y el archivo adjunto (si lo hay).
    - Luego, organiza toda esta información en un modelo y la muestra en una vista de detalles del ticket.

Vistas:

Las vistas utilizadas fueron desarrolladas con HTML + Bootstrap y JavaScript. A continuación, se muestran capturas de pantalla de las vistas principales:

Luego de haber iniciado sesión, se nos redirige a la siguiente vista:



Esta vista muestra la lista de los tickets. Ahora, si deseamos crear un nuevo ticket podemos hacerlo desde el botón o desde el acceso directo del menú de navegación.

Open new ticket

Email: carmen.tacuri@tecazuay.edu.ec

Cliente: Carmen Tacuri

Cédula: 0103533766

Help's topic

Infraestructura (Equipo, redes)

Details

Describe your problem

Mi computadora no enciende

File Edit View Insert Format

Subscript

Necesito ayuda, mi computadora no enciende y sí está conectada!!!

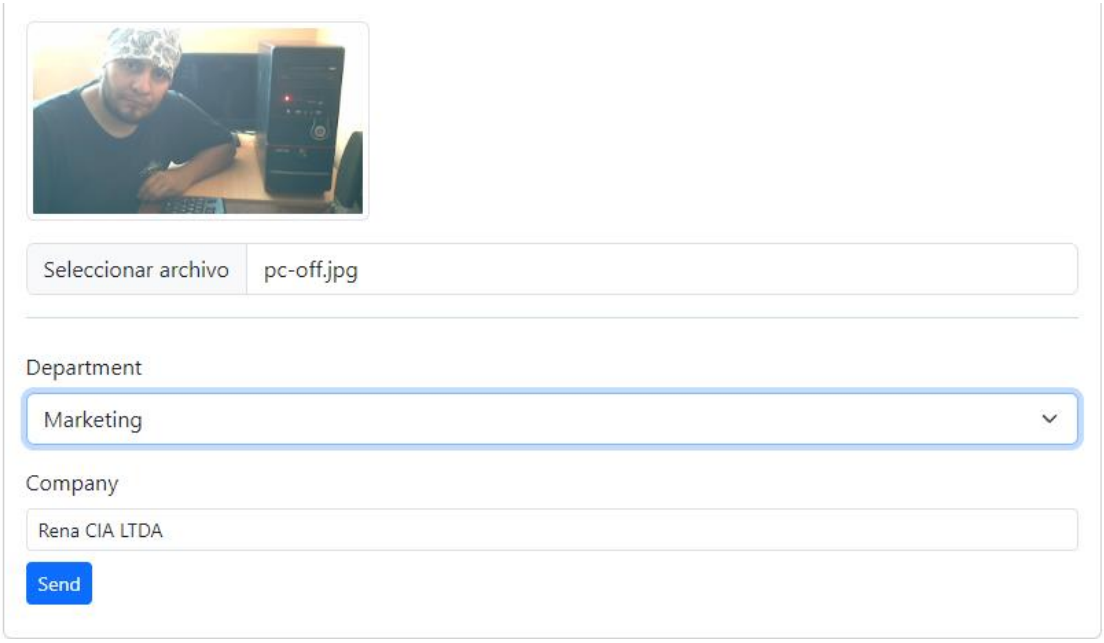
Por más que he intentado desconectando y conectando nuevamente, el problema persiste por lo que mi computadora no enciende.

No sé que hacer, necesito ayuda, por favor.

Att: Carmen Tacuri

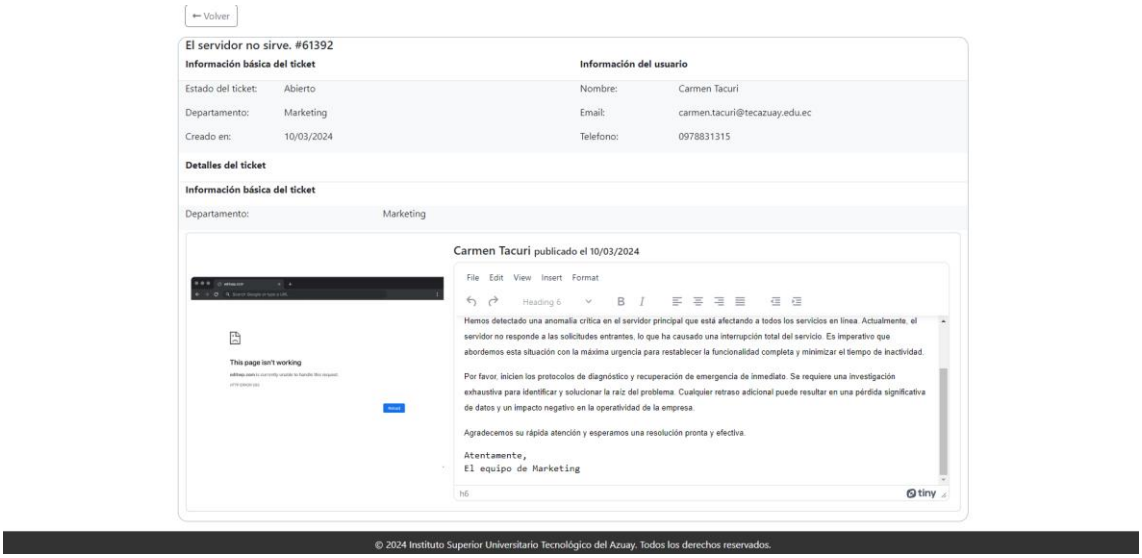
p › sub

tiny

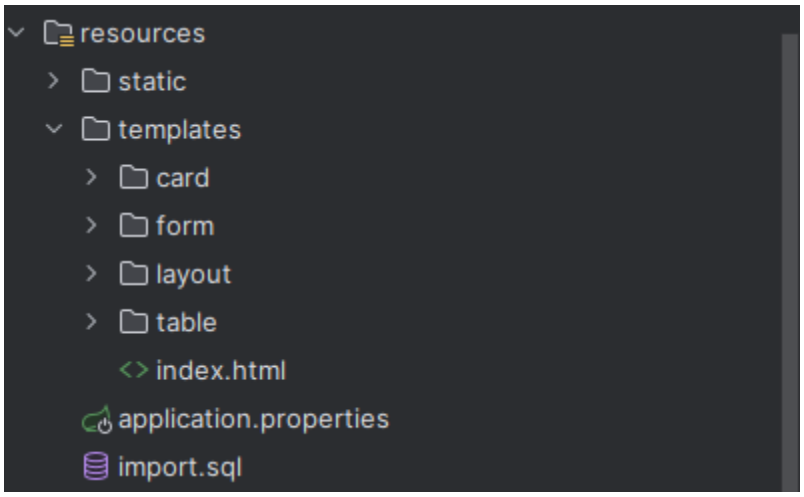


Ahora, gracias a este formulario podemos realizar la apertura de un nuevo ticket.

Y al abrir los detalles del ticket, podemos observar lo siguiente:



Nota: Si bien es cierto que hay más interfaces y funcionalidades, se describen únicamente estas de modo a que se apega a las instrucciones especificadas, además, las demás interfaces y funcionalidades están ligadas a la primera, es decir, mantienen una estructura bastante cercana a la del Ticket, tanto en funcionalidades como los tickets, exceptuando porque únicamente ticket es quien posee la opción para cargar fotos.



Como constancia de los elementos utilizados para las distintas interfaces. Adicionalmente, en cada una de ellas se utiliza un head, menú y footer compartido que fue definido en un layout.

```
<head th:fragment="head">
    <meta charset="UTF-8">
    <title th:text="${title}"></title>
    <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
    <link rel="stylesheet" th:href="@{https://use.fontawesome.com/releases/v5.12.1/css/all.css}">
    <script th:src="@{/umd/popper.min.js}"></script>
    <script th:src="@{/js/bootstrap.min.js}"></script>
    <script th:src="@{https://cdn.tiny.cloud/1/wh39i48d8y6k9j2cvhb8akvpaos8y91ig6qix86blwg7ujpi/tinymce/6/tinymce.min.js}" referrerpolicy="origin"></script>
</head>
```

De igual forma se definieron más fragmentos para compartir entre mi plantilla llamada “layout” y las demás plantillas desarrolladas.

```
<header th:fragment="header">
    <nav class="navbar navbar-expand-lg bg-body-tertiary">
        <div class="container-fluid">
            <a class="navbar-brand" href="/ticket/list-all-tickets">Support Tickets</a>

            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page" href="/ticket/list-all-tickets">List of tickets</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link active" href="/ticket/save-ticket">Create ticket</a>
                    </li>
                    <li class="nav-item dropdown">
                        <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown"
                           aria-expanded="false">
                            Additional options
                        </a>
                        <ul class="dropdown-menu active">
                            <li><a class="dropdown-item" href="/city/list-all-cities">Ciudades</a></li>
                            <li><hr class="dropdown-divider"></li>
                            <li><a class="dropdown-item" href="/company/list-all-companies">Empresas</a></li>
                            <li><hr class="dropdown-divider"></li>
                            <li><a class="dropdown-item" href="/department/list-all-departments">Departamentos</a></li>
                            <li><hr class="dropdown-divider"></li>
                            <li><a class="dropdown-item" href="/user/list-all-users">Usuarios</a></li>
                            <li><hr class="dropdown-divider"></li>
                            <li><a class="dropdown-item" href="/case/list-all-cases">Casos de ticket</a></li>
                        </ul>
                    </li>

                    <li class="nav-item ms-auto right">
                        <a class="nav-link active" aria-current="page" href="/user/logout">Log out</a>
                    </li>
                </ul>
            </div>
        </div>
```

```
    <div class="alert alert-success alert-dismissible fade show mt-2" role="alert" th:if="${success != null}">
        <strong th:text="${success}"></strong>
    </div>
    <div class="alert alert-danger alert-dismissible fade show mt-2" role="alert" th:if="${danger != null}">
        <strong th:text="${danger}"></strong>
    </div>
    <div class="alert alert-warning alert-dismissible fade show mt-2" role="alert" th:if="${warning != null}">
        <strong th:text="${warning}"></strong>
    </div>
    <div class="alert alert-info alert-dismissible fade show mt-2" role="alert" th:if="${info != null}">
        <strong th:text="${info}"></strong>
    </div>
</header>

<div class="container"></div>

<footer class="mt-3" th:fragment="footer" style="background-color: #333; color: white; text-align: center; padding: 10px 0;">
    <p>© 2024 Instituto Superior Universitario Tecnológico del Azuay. Todos los derechos reservados.</p>
    <p>Desarrollado el día 10. Diseñado con esmero por Lenin Córdova.</p>
</footer>
```

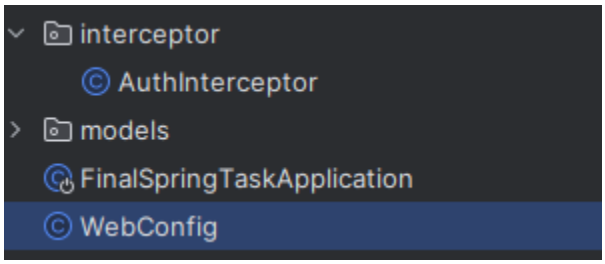
La manera en que se puede agregar este fragmento a mi template es de la siguiente manera:

```
<head th:replace="~{layout/layout :: head}"></head>
<body>
<header th:replace="~{layout/layout :: header}"></header>
```

```
<footer th:replace="~{layout/layout :: footer}"></footer>
```

Funcionalidad Adicional:

Para lograr lo de login y tener un control acerca de a donde ingresa y dónde no, se necesita controlar las peticiones mediante un “Interceptor”, esto da como resultado a la creación de 2 clases de configuración para mi aplicación.



Siendo “AuthInterceptor” el que intercepta las peticiones que se realizan dentro de mi aplicación a los endpoints que se han definido.

```java
@Component
public class AuthInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        HttpSession session = request.getSession();
        String requestSession = (String) session.getAttribute( name: "idCard");

        if (requestSession == null || requestSession.isEmpty()) {
            response.sendRedirect( location: "/user/login");
            return false; // No continuar con el flujo hacia el controlador
        }

        return true; // Continuar con el flujo hacia el controlador si la sesión es válida
    }
}
```

- implements HandlerInterceptor: Al implementar esta interfaz, AuthInterceptor debe definir los métodos preHandle, postHandle, y afterCompletion. En tu caso, solo se ha implementado preHandle.
- preHandle: Este método se llama justo antes de que la solicitud sea manejada por un controlador. Es utilizado para realizar operaciones como autenticación o autorización.
- Dentro del método preHandle:
    - HttpSession session = request.getSession();: Obtiene la sesión actual de la solicitud HTTP.
    - String requestSession = (String) session.getAttribute("idCard");: Intenta obtener un atributo llamado “idCard” de la sesión. Este atributo probablemente se utiliza para verificar si el usuario ha iniciado sesión.
    - if (requestSession == null || requestSession.isEmpty()): Comprueba si el atributo “idCard” no existe o está vacío, lo cual indicaría que el usuario no está autenticado.
    - response.sendRedirect("/user/login");: Si el usuario no está autenticado, redirige la solicitud a la ruta “/user/login”, que probablemente es la página de inicio de sesión.

- return false;: Detiene el flujo de la solicitud, evitando que llegue al controlador.
- return true;: Si el usuario está autenticado (es decir, “idCard” está presente y no está vacío), permite que la solicitud continúe hacia el controlador.

Por lo que, AuthInterceptor es un componente que asegura que el usuario esté autenticado antes de permitirle acceder a ciertas rutas en la aplicación. Si el usuario no está autenticado, se le redirige a la página de inicio de sesión.

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Autowired
    AuthInterceptor authInterceptor;

    2 usages
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(authInterceptor).addPathPatterns("/**")
                .excludePathPatterns("/user/login")
                .excludePathPatterns("/static/**")
                .excludePathPatterns("/css/**") // Excluir CSS
                .excludePathPatterns("/js/**") // Excluir JavaScript
                .excludePathPatterns("/images/**"); // Excluir imágenes;
    }
}
```

- addInterceptors(InterceptorRegistry registry): Este método se sobrescribe para registrar interceptores personalizados que actuarán sobre las solicitudes entrantes antes de que sean manejadas por los controladores.
- Dentro del método addInterceptors:
  - registry.addInterceptor(authInterceptor): Registra el AuthInterceptor para que sea aplicado a las solicitudes.
  - .addPathPatterns("/**"): Indica que el interceptor se aplicará a todas las rutas de la aplicación.
  - .excludePathPatterns("/user/login"): Excluye la ruta /user/login del interceptor, lo que significa que las solicitudes a esta ruta no serán interceptadas y, por lo tanto, no se requerirá autenticación.
  - .excludePathPatterns("/static /**"): Excluye todas las rutas bajo /photo-public/ del interceptor, permitiendo el acceso público a estos recursos.
  - .excludePathPatterns("/css/**"): Excluye todas las rutas bajo /css/ para que los archivos CSS sean accesibles sin autenticación.
  - .excludePathPatterns("/js/**"): Excluye todas las rutas bajo /js/ para que los archivos JavaScript sean accesibles sin autenticación.

- .excludePathPatterns("/images/**"): Excluye todas las rutas bajo /images/ para que las imágenes sean accesibles sin autenticación.

Por lo que, esta configuración asegura que AuthInterceptor se aplique a todas las rutas excepto a las especificadas en las exclusiones, permitiendo así que ciertos recursos estáticos y la página de inicio de sesión sean accesibles sin necesidad de autenticación. Esto es útil para permitir que los usuarios accedan a recursos públicos y para iniciar sesión sin ser redirigidos por el interceptor.

Conclusiones:

Spring y Thymeleaf representan una sinergia tecnológica que potencia el desarrollo de aplicaciones web en Java. Spring, con su robusto ecosistema, ofrece una plataforma sólida para la construcción de aplicaciones complejas y escalables, mientras que Thymeleaf complementa esta robustez al proporcionar una manera intuitiva y eficiente de construir interfaces de usuario ricas y dinámicas.

Esta combinación no solo mejora la productividad del desarrollador, sino que también promueve prácticas de desarrollo coherentes y mantenibles, lo que resulta en aplicaciones web que son tanto funcionales como estéticamente agradables. Además, la naturaleza de código abierto de ambas tecnologías fomenta una comunidad activa de desarrolladores que contribuyen constantemente con mejoras y soporte, asegurando que las aplicaciones construidas con Spring y Thymeleaf estén a la vanguardia de la innovación tecnológica.