Epic Event

Customer Relational Management

THIERRY Edwin

1. Introduction et context

2. Mise en place des technologies

3. Configuration de l’environnement

4. Structure et fonctionnalités

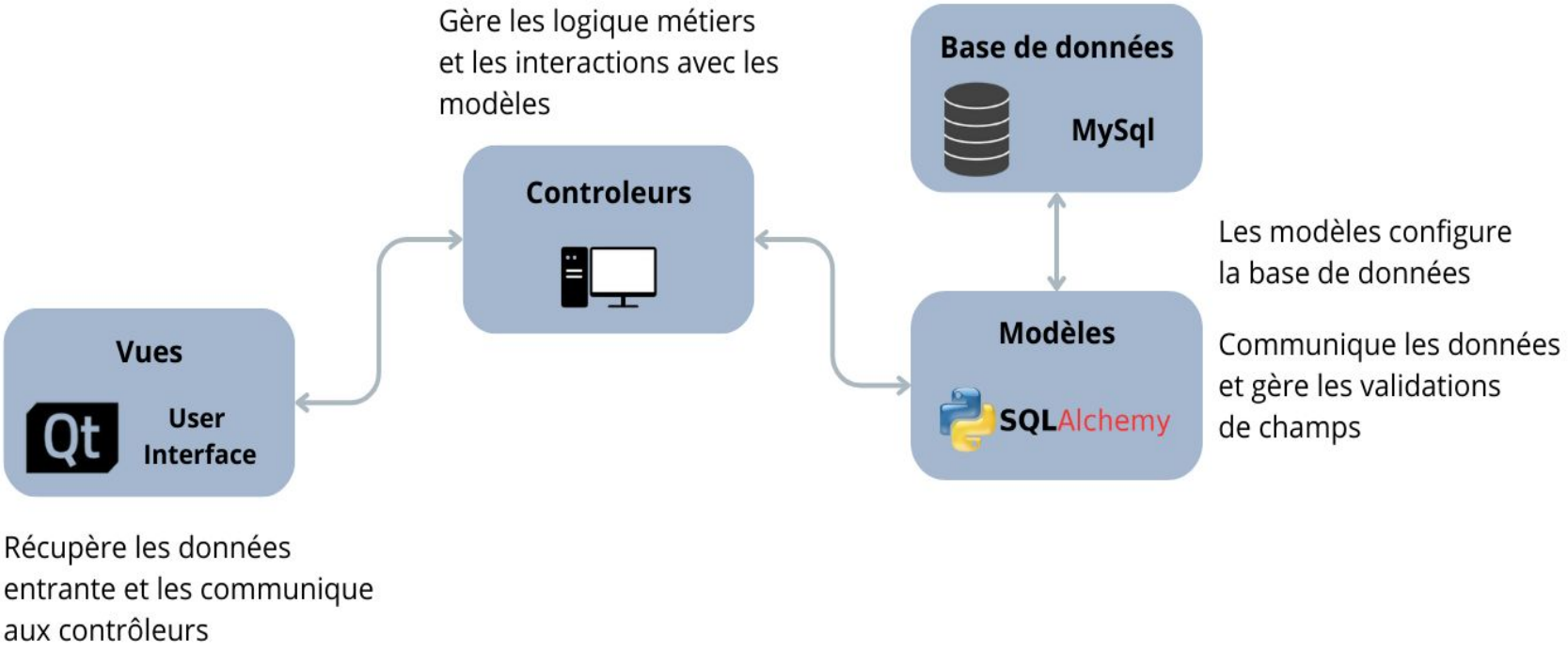
5. Couverture de test

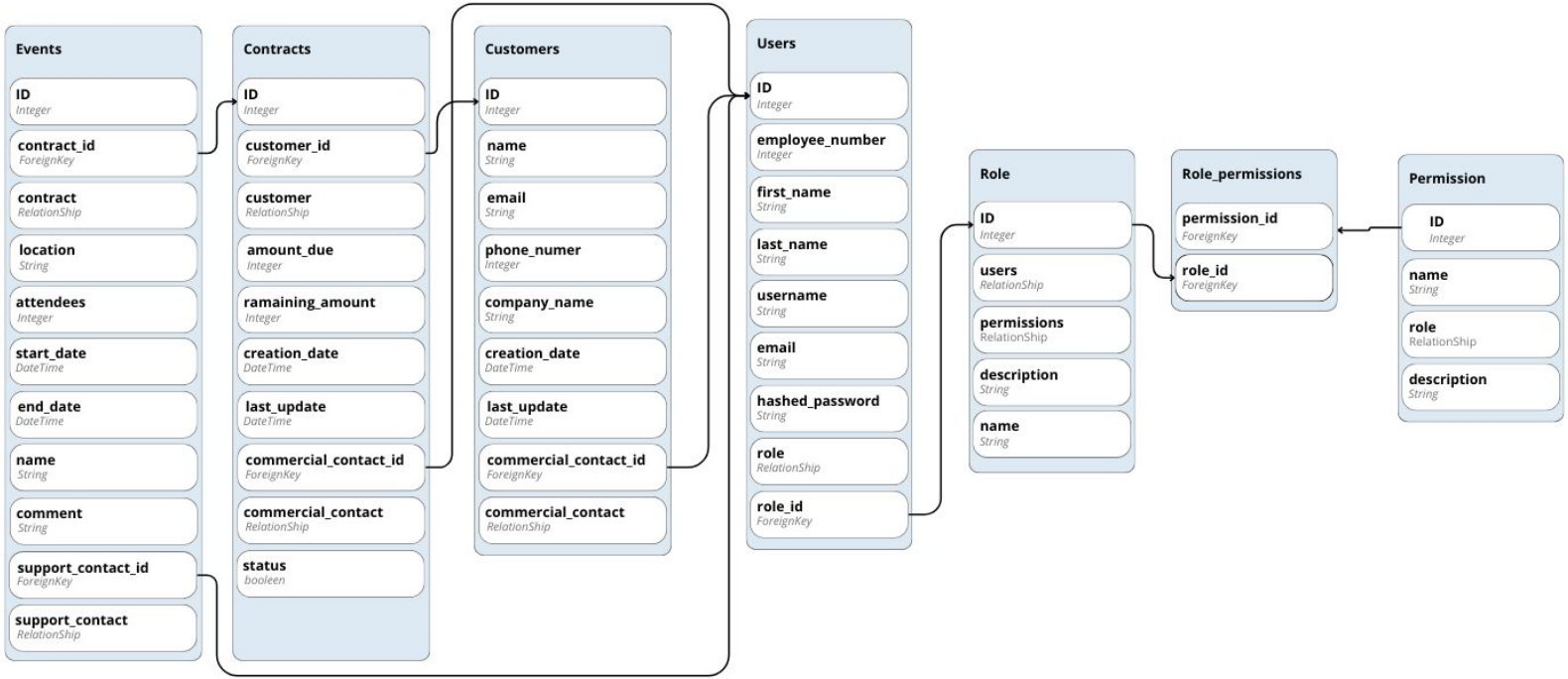
1. Introduction

- Mise en place d’une Base de données
- Manipulation sécurisé des informations
- Sécurités et validations
- Suivis des erreurs et maintenabilité

Système complet de gestion des relations clients et des contrats d’organisation événementiel

2. Mise en place




Events
ID
Integer
contract_id
ForeignKey
contract
RelationShip
location
String
attendees
Integer
start_date
DateTime
end_date
DateTime
name
String
comment
String
support_contact_id
ForeignKey
support_contact
RelationShip
Contracts
ID
Integer
customer_id
ForeignKey
customer
RelationShip
amount_due
Integer
ramaining_amount
Integer
creation_date
DateTime
last_update
DateTime
commercial_contact_id
ForeignKey
commercial_contact
RelationShip
status
booleen
Customers
ID
Integer
name
String
email
String
phone_numer
Integer
company_name
String
creation_date
DateTime
last_update
DateTime
commercial_contact_id
ForeignKey
commercial_contact
RelationShip
Users
ID
Integer
employee_number
Integer
first_name
String
last_name
String
username
String
email
String
hashed_password
String
role
RelationShip
role_id
ForeignKey
Role
ID
Integer
users
RelationShip
permissions
RelationShip
description
String
name
String
Role_permissions
permission_id
ForeignKey
role_id
ForeignKey
Permission
ID
Integer
name
String
role
RelationShip
description
String

```python
class User(Base, BaseModelMixin):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    employee_number = Column(Integer, nullable=False)
    email = Column(String(100), unique=True, index=True)
    first_name = Column(String(100), unique=False, index=True)
    last_name = Column(String(100), unique=False, index=True)
    username = Column(String(100), unique=True, index=True)
    hashed_password = Column(String(128))
    is_active = Column(Boolean, default=True)
    is_admin = Column(Boolean, default=False)

    # ForeignKey to Role
    role_id = Column(Integer, ForeignKey("roles.id"))
    # RelationShip with Role
    role = relationship("Role", back_populates="users")
```

- Champs du modèle
- Relation avec les rôles

```python
def set_password(self, password):
    # Salage et hashage du password avant de l'enregistrer
    salt = bcrypt.gensalt()
    self.hashed_password = bcrypt.hashpw(password.encode("utf-8"), salt).decode(
        "utf-8"
    )

def check_password(self, password):
    # Vérifie le password avec sa valeur hashée dans la db
    return bcrypt.checkpw(
        password.encode("utf-8"), self.hashed_password.encode("utf-8")
    )
```

- Méthode de Hachage et salage du mot de passe

```python
@property
def full_name(self):
    return f"{self.first_name} {self.last_name}"

@validates("first_name", "last_name")
def validate_name(self, key, name):
    name = name.replace(" ", "")
    if not re.match("^[a-zA-ZÀ-ÿ'-]+$", name):
        raise ValueError(
            f"Invalid {key}. Only letters, hyphens, and apostrophes are allowed."
        )
    return name

@validates("email")
def validate_email(self, key, email):
    email_regex = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
    if not re.match(email_regex, email):
        raise ValueError("Invalid email format")
    return email

@validates("username")
def validate_username(self, key, username):
    if not username or not re.match("^[a-zA-Z0-9_.-]+$", username):
        raise ValueError("Invalid username format")
    return username

@validates("employee_number")
def validate_employee_number(self, key, employee_number):
    employee_number_str = str(employee_number)
    if not employee_number_str.isdigit() or len(employee_number_str) > 3:
        raise ValueError(
            "Employee number must be numeric and less than or equal to 3 digits"
        )
    return employee_number
```

3. Initialisation

- Cryptage du mot de passe de la base de données
- Automatisation de l'initialisation de la base de données

Le système est livré avec les tables initialisé, ainsi que les rôles et les permissions

Un utilisateur manageur est alors créer pour une première utilisation

```python
def setup_env_file():
    env_file_path = Path(".env")

    # Si le fichier .env n'existe pas
    if not env_file_path.exists():
        print("No .env file found, Please enter a database password")
        db_password = getpass("Database password: ")
        # Toujours générer une clé secrète si elle n'existe pas dans l'environnement
        secret_key = os.getenv("SECRET_KEY")
        if not secret_key:
            secret_key = generate_key().decode()
        # Chiffrer le mot de passe
        encrypted_password = encrypt_password(db_password, secret_key)
        # Écriture dans le fichier .env
        with open(".env", "w") as env_file:
            env_file.write(f"SECRET_KEY={secret_key}\n")
            env_file.write(f"DB_PASSWORD_ENCRYPTED={encrypted_password}\n")
        print("Welcome to Epic Event CRM")
        load_dotenv()

    else:
        load_dotenv()
        if not os.getenv("DB_PASSWORD_ENCRYPTED"):
            print("No password found in .env, Please enter a database password")
            db_password = getpass("Database password: ")

            secret_key = os.getenv("SECRET_KEY")
            if not secret_key:
                secret_key = generate_key().decode()
            encrypted_password = encrypt_password(db_password, secret_key)
            with open(
                ".env", "a"
            ) as env_file:  # Ouvrir le fichier en mode ajout si déjà existant
                env_file.write(f"DB_PASSWORD_ENCRYPTED={encrypted_password}\n")
            print("Encrypted password written in .env file")
            print("Welcome to Epic Event CRM")
```

- Demande le mot de passe de la DB si aucun mot de passe n’est trouvé
- Encrypte le mot de passe grâce à une secret key généré

```python
def configure_database():

    os.environ.pop("DB_PASSWORD_ENCRYPTED", None)
    os.environ.pop("SECRET_KEY", None)

    load_dotenv()

    encrypted_password = os.getenv("DB_PASSWORD_ENCRYPTED")
    secret_key = os.getenv("SECRET_KEY")

    db_password = decrypt_password(encrypted_password, secret_key)
    database_url = f"mysql+mysqldb://Admin:{db_password}@localhost:3306/epic_event_crm"

    # SQLAlchemy setup
    engine = create_engine(database_url, echo=False)
    SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

    return SessionLocal(), engine


# Declarativa Base for SqlAlchemy models
Base = declarative_base()
```

4. Fonctionnalités

Commercial	Support	Management
● Créer des clients ● Mettre à jour les clients dont ils sont responsables. ● Modifier/mettre à jour les contrats des clients dont ils sont responsables. ● Filtrer l'affichage des contrats ● Créer un événement	● Filtrer l'affichage des événements, par exemple : afficher uniquement les événements qui leur sont attribués. ● Mettre à jour les événements dont ils sont responsables	● Créer, mettre à jour et supprimer des collaborateurs ● Créer et modifier tous les contrats. ● Filtrer l'affichage des événements ● Modifier des événements

```python
def require_permission(permission_name):
    def decorator(func):
        @wraps(func)
        def wrapper(self, *args, **kwargs):
            if not self.authenticated_user.has_permission(permission_name):
                raise PermissionError(
                    f"You do not have permission to {permission_name}."
                )
            return func(self, *args, **kwargs)

        return wrapper

    return decorator
```

- Vérification des permissions de l'utilisateur connecté

```python
def is_authenticated_user(func):
    @wraps(func)
    def wrapper(self, *args, **kwargs):
        if not self.authenticated_user:  # Vérifie si l'utilisateur est authentifié
            print("is authenticated user permission")
            raise PermissionError("You do not have permission")

        return func(self, *args, **kwargs)

    return wrapper
```

- Vérification de l'utilisateur connecté

```python
def view_authenticated_user(func):
    @wraps(func)
    def wrapper(self, *args, **kwargs):
        if not self.controller.authenticated_user:
            print("view permission")
            raise PermissionError("You do not have permission")

        print(f" AUTH :: {self.controller.authenticated_user.first_name}")
        return func(self, *args, **kwargs)
```

- Vérification de l'utilisateur connecté pour l'interface

```python
@decorate_all_methods(is_authenticated_user)
class CommercialController(MainController):
    def __init__(self, session, authenticated_user, login_controller):
        self.session = session
        self.authenticated_user = authenticated_user
        self.login_controller = login_controller
```

- initialisation de la session, ainsi que de l'utilisateur connecté

```python
    @require_permission("create_customer")
    def create_customer(self, **customer_data):
        """Add new customer with customer data"""
```

- Ajout du décorateur avec la permission correspondante

```python
        try:
            new_customer = Customer(
                name=customer_data["name"],
                email=customer_data["email"],
                phone_number=customer_data["phone_number"],
                company_name=customer_data["company_name"],
                commercial_contact_id=self.authenticated_user.id,
            )
            self.session.add(new_customer)
            self.session.commit()
            return new_customer
        except Exception as e:
            self.session.rollback()
            raise ValueError(f"An error occurred while creating the customer: {str(e)}")
```

- Bloc Try/Except pour catch les erreurs (Sentry)

CONCLUSION

- Choix des technologies cohérent
- Sécurités renforcée
- Séparation des responsabilités
- Système de journalisation
- Engagement en maintenabilité
- Couverture de test