EPIC Events

Contexte

Epic Events est une entreprise qui organise des événements (fêtes, réunions professionnelles, manifestations hors les murs) pour ses clients. Nous souhaitons développer un logiciel CRM (Customer Relationship Management) pour améliorer notre travail.

Le logiciel CRM permet de collecter et de traiter les données des clients et de leurs événements, tout en facilitant la communication entre les différents pôles de l'entreprise.

Comment Epic Events fonctionne

Nos équipes opérationnelles sont divisées en trois départements :

- le département commercial ;
- le département support ;
- le département gestion.

Python :

```python
class EpicEventRole(Enum):
    collaborator = "collaborator"
    commercial = "commercial"
    support = "support"
    gestion = "gestion"
```

MySQL

```
CREATE_ROLE_COLLABORATOR = "CREATE ROLE 'collaborator'"
CREATE_ROLE_COMMERCIAL = "CREATE ROLE 'commercial'"
CREATE_ROLE_SUPPORT = "CREATE ROLE 'support'"
CREATE_ROLE_GESTION = "CREATE ROLE 'gestion'"

DROP_ROLE_COLLABORATOR = "DROP ROLE IF EXISTS 'collaborator'"
DROP_ROLE_COMMERCIAL = "DROP ROLE IF EXISTS 'commercial'"
DROP_ROLE_SUPPORT = "DROP ROLE IF EXISTS 'support'"
DROP_ROLE_GESTION = "DROP ROLE IF EXISTS 'gestion'"
```

Les commerciaux démarchent les clients. Ils créent et mettent à jour leurs profils sur la plateforme. Lorsqu'un client souhaite organiser un événement, un collaborateur du département gestion crée un contrat et l'associe au client.

```
ADD_PRIVILEGES_COMMERCIAL_ROLE = (
    "GRANT INSERT, UPDATE ON epicevent.client TO 'commercial'"
)
ADD_PRIVILEGES_COMMERCIAL_ROLE_2 = (
    "GRANT INSERT, UPDATE ON epicevent.contract TO 'commercial'"
)
ADD_PRIVILEGES_GESTION_ROLE_2 = (
    "GRANT INSERT,UPDATE ON epicevent.contract TO 'gestion';"
)
```

```
class CreateClientView(CreateView):
    name_display = "Créer un nouveau client"

    view_authorization = [authorization.IsCommercial]
```

```
class CreateContractView(CreateView):
    name_display = "Créer un contrat"

    view_authorization = [authorization.IsNotSupport]
    context_requirements = ["client"]
```

Une fois le contrat signé, le commercial crée l'événement dans la plateforme et le département gestion désigne un membre du département support qui sera responsable de l'organisation et du déroulé de l'événement.

```
ADD_PRIVILEGES_COMMERCIAL_ROLE_3 = "GRANT INSERT ON epicevent.event TO 'commercial'"
ADD_PRIVILEGES_GESTION_ROLE_3 = (
    "GRANT UPDATE (support_contact, last_update) ON epicevent.event TO 'gestion';"
)
```

```
class EventCreateView(CreateView):
    name_display = "Créer un évènement"

    context_requirements = ["contract"]

    Archy
    @classmethod
    def are_requirements_meet(cls, context) -> bool:
        if not super().are_requirements_meet(context):
            return False

        contract = context.get_in_context("contract")
        return contract.statut.lower() != ContractStatus.to_sign.value.lower()
```

Les clients

Un client est identifié par les informations suivantes :

Information	Exemple
Nom complet	Kevin Casey
Email	kevin@startup.io
Téléphone	+678 123 456 78
Nom de l'entreprise	Cool Startup LLC
Date de création (= premier contact)	18 avril 2021
Dernière mise à jour/contact	29 mars 2023
Contact commercial chez Epic Events	Bill Boquet

```python
@dataclasses.dataclass
class Client:
    full_name: str
    company_name: str
    email: str
    phone: str
    creation_date: str = datetime.now().strftime("%Y-%m-%d")
    last_update: str = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    client_id: int = -1
    collaborator_id: int = -1
```

```python
2 usages  Archy
class ClientController(BaseController):
    views = [ClientView, UserOwnClientView, CreateClientView]
    serializers = ClientSerializer

    table = "client"
    model = Client
    form = ClientCreationForm
```

```sql
CREATE_CLIENT_QUERY = """
CREATE TABLE client (
  client_id INT AUTO_INCREMENT PRIMARY KEY,
  collaborator_id INT,
  full_name VARCHAR(40) NOT NULL,
  company_name VARCHAR(40) NOT NULL,
  email VARCHAR(30) NOT NULL,
  phone VARCHAR(30) NOT NULL,
  creation_date DATE NOT NULL,
  last_update DATETIME,
  is_active BOOLEAN DEFAULT TRUE
  );
 """
```

```sql
CLIENT_FULL_VIEW = """
CREATE OR REPLACE VIEW epicevent.list_client AS
SELECT
  client_id AS 'Client ID',
  company_name AS 'Société',
  full_name AS 'Intérlocuteur',
  email AS 'adresse mail',
  phone AS 'téléphone',
  creation_date AS 'client depuis',
  last_update AS 'Dernière MàJ',
  collaborator_id AS 'Commercial associé'
FROM client;
"""
```

Les contrats

Un contrat contient :

- un identifiant unique ;
- les informations sur le client ;
- le contact commercial pour le contrat (= le commercial associé au client) ;
- le montant total du contrat ;
- le montant restant à payer ;
- la date de création du contrat ;
- le statut du contrat (= si le contrat a été signé).

```python
@dataclasses.dataclass
class Contract:
    cost: float
    balance: float
    statut: int = 1
    creation_date: str = datetime.now().strftime("%Y-%m-%d")
    last_update: str = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    contract_id: int = -1
    client_id: int = -1

class ContractController(BaseController):
    views = [
        ContractView,
        UserClientContractView,
        CreateContractView,
        UnsignedContract,
        UnPayedContract,
    ]
    serializers = ContractSerializer

    table = "contract"
    model = Contract
    form = ContractCreationForm
```

```python
CREATE_CONTRACT_QUERY = """
CREATE TABLE contract (
  contract_id INT AUTO_INCREMENT PRIMARY KEY,
  client_id INT,
  cost FLOAT,
  balance FLOAT,
  statut VARCHAR(20),
  creation_date DATE,
  last_update DATETIME,
  is_active BOOLEAN DEFAULT TRUE
  );
 """
```

```python
CONTRACT_FULL_VIEW = """
CREATE OR REPLACE VIEW epicevent.list_contract AS
SELECT
  contract_id AS `Contrat ID`,
  client_id AS 'Client ID',
  cost AS 'Cout',
  balance AS 'Balance',
  statut AS 'Statut',
  creation_date AS 'Sous contrat depuis',
  last_update AS 'Dernière MàJ'
FROM contract;
"""
```

Les événements

À l'heure actuelle, le département support utilise une feuille Excel pour suivre les événements. Voici deux exemples d'événements :

	A	B
1	**John Ouick Wedding**	
2	Event ID	109
3	Contract ID	652
4	Client name	John Ouick
5	Client contact	john.ouick@gmail.com +1 234 567 8901
6	Event date start	4 Jun 2023 @ 1PM
7	Event date end	5 Jun 2023 @ 2AM
8	Support contact	Kate Hastroff
9	Location	53 Rue du Château, 41120 Candé-sur-Beuvron, France
10	Attendees	75
11	Notes	Wedding starts at 3PM, by the river. Catering is organized, reception starts at 5PM. Kate needs to organize the DJ for after party.

```python
@dataclasses.dataclass
class Event:
    name: str
    event_start: str
    event_end: str
    location: str
    attendees: int
    notes: str
    creation_date: str = datetime.now().strftime("%Y-%m-%d")
    last_update: str = datetime.now().strftime("%Y-%m-%d")
    event_id: int = -1
    contract_id: int = -1
    support_contact: Optional[int] = None
```

```python
class EventController(BaseController):
    views = [
        EventView,
        UnassignedEventView,
        UserAssignedEventView,
        EventCreateView,
        CommercialClientEventView,
    ]
    serializers = EventSerializer

    model = Event
    form = EventCreationForm
    table = "event"
```

	A	B
1	**Lou Bouzin General Assembly**	
2	Event ID	1023
3	Contract ID	412
4	Client name	Lou Bouzin
5	Client contact	Jacky Blangier jacky@loubouzin.grd + 666 12345
6	Event date start	5 May 2023 @ 3PM
7	Event date end	5 May 2023 @ 5PM
8	Support contact	Aliénor Vichum
9	Location	Salle des fêtes de Mufflins
10	Attendees	200
11	Notes	Assemblée générale des actionnaires (~200 personnes).

```
CREATE_EVENT_QUERY = """
CREATE TABLE event (
  event_id INT AUTO_INCREMENT PRIMARY KEY,
  contract_id INT,
  support_contact INT DEFAULT NULL,
  name VARCHAR(30),
  event_start DATETIME,
  event_end DATETIME,
  location VARCHAR(50),
  attendees INT,
  notes TINYTEXT,
  creation_date DATE,
  last_update DATETIME,
  is_active BOOLEAN DEFAULT TRUE
);
"""
```

```
EVENT_FULL_VIEW = """
CREATE OR REPLACE VIEW epicevent.list_event AS
SELECT
  contract_id AS `Contrat ID`,
  event_id AS 'Event ID',
  support_contact AS 'Contact de support',
  name AS 'Événement',
  event_start AS 'Début',
  event_end AS 'Fin',
  location AS 'Lieu',
  attendees AS 'Participants',
  notes AS 'Notes' ,
  last_update AS 'Dernière MàJ'
FROM event;
"""
```

Besoins individuels : équipe de gestion

- Créer, mettre à jour et supprimer des collaborateurs dans le système CRM.
- Créer et modifier tous les contrats.
- Filtrer l'affichage des événements, par exemple : afficher tous les événements qui n'ont pas de « support » associé.
- Modifier des événements (pour associer un collaborateur support à l'événement).

```
ADD_PRIVILEGES_GESTION_ROLE_3 = (
    "GRANT UPDATE (support_contact, last_update) ON epicevent.event TO 'gestion';"
)

GESTION_UNASSIGNED_EVENT = """
CREATE OR REPLACE VIEW epicevent.unassigned_event AS
SELECT * FROM list_event
WHERE `Contact de support` IS NULL;
"""
```

```
class CollaboratorView(BaseView):
    view_name = "list_collaborator"
    name_display = "Voir mes collaborateurs"

    view_authorization = [authorization.IsGestion]
    select_authorization = [authorization.IsGestion]
    delete_authorization = [authorization.IsGestion]


2 usages  Archy
class CreateCollaboratorView(CreateView):
    name_display = "Créer un collaborateur"

    view_authorization = [authorization.IsGestion]
```

```
class CreateContractView(CreateView):
    name_display = "Créer un contrat"

    view_authorization = [authorization.IsNotSupport]
    update_authorization = [authorization.IsNotSupport]
    context_requirements = ["client"]
```

```
class UnassignedEventView(EventView):
    allow_select = True

    view_name = "unassigned_event"
    name_display = "Voir les événements non assigné"

    view_authorization = [authorization.IsGestion]
```

Besoins individuels : équipe commerciale

- Créer des clients (le client leur sera automatiquement associé).
- 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, par exemple : afficher tous les contrats qui ne sont pas encore signés, ou qui ne sont pas encore entièrement payés.
- Créer un événement pour un de leurs clients qui a signé un contrat.

```python
class CreateClientView(CreateView):
    name_display = "Créer un nouveau client"

    view_authorization = [authorization.IsCommercial]

class UserOwnClientView(ClientView):
    view_name = "commercial_own_client"
    name_display = "Voir mes clients"

    view_authorization = [authorization.IsCommercial]
    select_authorization = [authorization.IsCommercial]
    update_authorization = [authorization.IsCommercial]
    delete_authorization = [authorization.Forbidden]
```

```python
AUTO_ASSIGN_COMMERCIAL_TO_CLIENT = """
CREATE TRIGGER auto_assign_commercial_to_client BEFORE INSERT ON client
FOR EACH ROW
SET new.collaborator_id = get_collaborator_id();
"""
```

```python
class UserClientContractView(ContractView):
    view_name = "commercial_client_contracts"
    name_display = "Voir les contrats de mes clients"

    view_authorization = [authorization.IsCommercial]
    select_authorization = [authorization.IsCommercial]


2 usages  Archy
class UnsignedContract(UserClientContractView):
    name_display = "Voir les contrat pas encore signé."

    filter = f"`Statut` = '{ContractStatus.to_sign.value}'"
    context_requirements = [UserClientContractView]


2 usages  Archy
class UnPayedContract(UserClientContractView):
    name_display = "Voir les contrat pas encore payé."

    filter = f"`Statut` = '{ContractStatus.to_pay.value}'"
    context_requirements = [UserClientContractView]
```

```python
class EventCreateView(CreateView):
    name_display = "Créer un évènement"

    context_requirements = ["contract"]

    Archy
    @classmethod
    def are_requirements_meet(cls, context) -> bool:
        if not super().are_requirements_meet(context):
            return False

        contract = context.get_in_context("contract")
        return contract.statut.lower() != ContractStatus.to_sign.value.lower()
```

Besoins individuels : équipe support

- Filtrer l'affichage des événements, par exemple : afficher uniquement les événements qui leur sont attribués.
- Mettre à jour les événements dont ils sont responsables.

```
SUPPORT_ASSIGNEE_EVENT = """
CREATE OR REPLACE VIEW epicevent.user_assigned_event AS
SELECT * FROM list_event
WHERE `Contact de support` = get_collaborator_id();
"""
```

```
class UserAssignedEventView(UnassignedEventView):
    view_name = "user_assigned_event"
    name_display = "Voir les événements qui me sont assignés"

    view_authorization = [authorization.IsSupport]
    select_authorization = [authorization.IsSupport]
    update_authorization = [authorization.IsSupport]
```