📋 Documentation Complète - Système de Sondages Distribué

1️⃣ PRÉSENTATION GÉNÉRALE DU PROJET

🎯 Objectif Principal

Créer une **plateforme de sondages moderne et distribuée** permettant aux utilisateurs de:

- **Créer** des sondages avec questions personnalisées
- **Voter** sur les options proposées
- **Consulter** les résultats en temps réel avec visualisation graphique

📊 Type de Projet

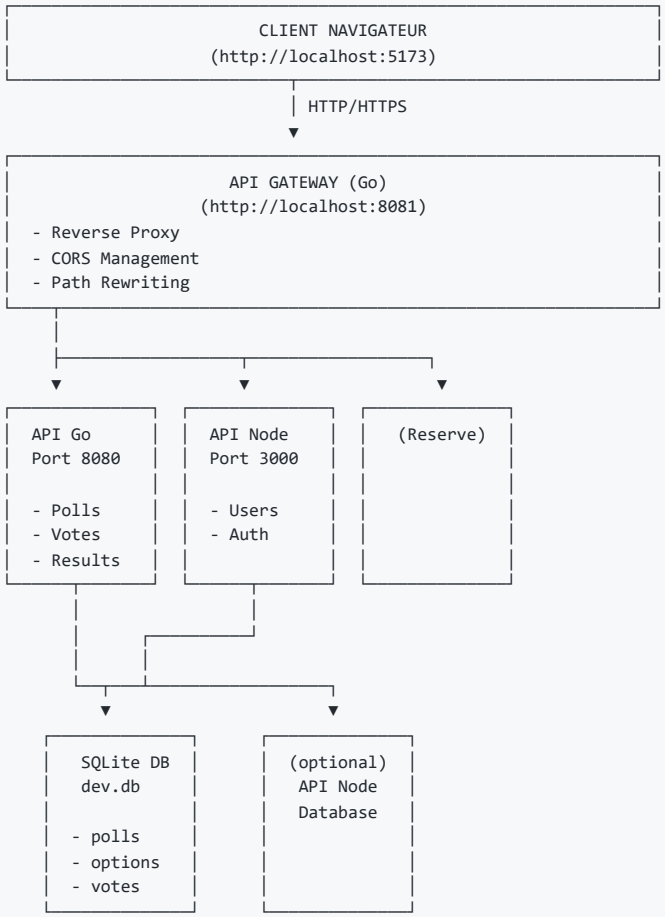
- **Full-Stack Web Application**
- **Architecture Microservices** avec API Gateway
- **Containerisée** (Docker & Docker Compose)
- **Base de données relationnelle** (SQLite)

👥 Public Cible

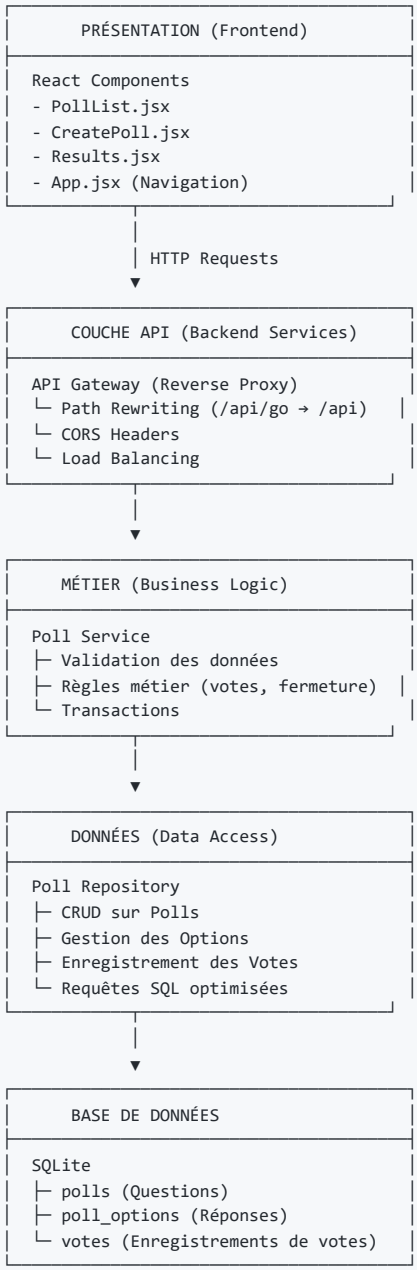
Organisations, équipes de travail, ou communautés nécessitant un outil de sondage rapide et fiable.

2️⃣ ARCHITECTURE SYSTÈME COMPLÈTE

🏗️ Vue d'ensemble globale



📦 Architecture en Couches



3️⃣ STACK TECHNOLOGIQUE DÉTAILLÉ

Frontend (Client-React)

Composant	Version	Rôle
React	18.2.0	Framework UI principal
Vite	4.0.0	Bundler et serveur dev (< 100ms HMR)
@tanstack/react-query	4.22.4	Gestion état async, cache API
axios	1.2.3	Client HTTP
CSS-in-JS	Custom	Styling avec CSS custom properties

Pourquoi ce choix:

- React: Écosystème mature, composants réutilisables
- Vite: Bundler ultra-rapide, meilleure DX que Webpack
- React Query: Gestion automatique du cache, refetch intelligent
- axios: Simplicité d'utilisation, interceptors pour erreurs

Backend - API Go

Composant	Version	Rôle
Go	1.21	Langage compilé, haute performance
Gin	Latest	Framework web léger et rapide
database/sql	Built-in	Requêtes SQL type-safe
sqlite3	cgo enabled	Driver SQLite

Pourquoi ce choix:

- Go: Compilation rapide, binaire unique, concurrence native
- Gin: Framework minimaliste, routage efficace, middleware simple
- SQLite: Base de données fichier, zéro configuration, parfait pour dev/test

Conteneurisation & Orchestration

Outil	Version	Usage
Docker	Latest	Isolation des services
Docker Compose	Latest	Orchestration locale multi-conteneur

Services conteneurisés:

1. `api-golang` - API sondages (port 8080)
2. `api-node` - API utilisateurs (port 3000, optionnel)
3. `api-gateway` - Reverse proxy (port 8081)
4. `client-react` - Frontend Vite (port 5173)

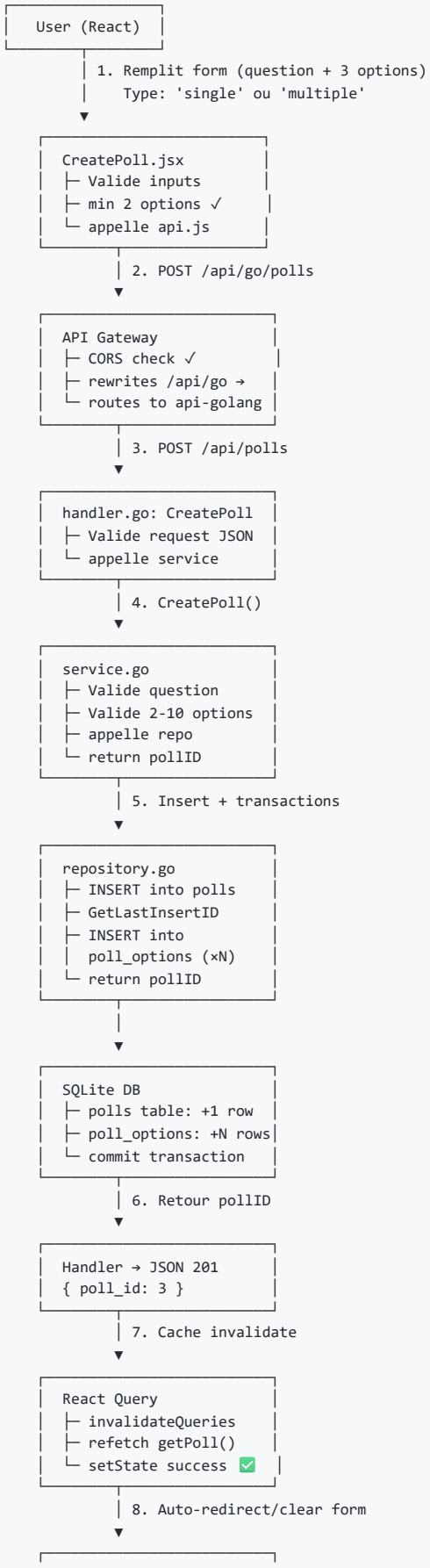
4️⃣ STRUCTURE DES RÉPERTOIRES

```
Devops-Project/
├── api-golang/                          # Backend Go
│   ├── Dockerfile                       # Image Docker
│   ├── go.mod & go.sum                  # Dépendances Go
│   ├── main.go                          # Entry point, routes
│   │
│   ├── database/
│   │   ├── db.go                        # Initialisation DB
│   │   └── 005_team5_polls.sql          # Schéma initial
│   │
│   ├── poll/                            # Package métier
│   │   ├── handler.go                   # HTTP handlers
│   │   │   ├ CreatePoll(c *gin.Context)
│   │   │   ├ GetPoll(c *gin.Context)
│   │   │   ├ GetPolls(c *gin.Context)
│   │   │   ├ Vote(c *gin.Context)
│   │   │   ├ GetResults(c *gin.Context)
│   │   │   └ ClosePoll(c *gin.Context)
│   │   │
│   │   ├── service.go                   # Logique métier
│   │   │   ├ CreatePoll()  [validation + insert]
│   │   │   ├ Vote()        [règles métier]
│   │   │   ├ GetPoll()     [fetch + options]
│   │   │   ├ GetAllPolls() [list avec options]
│   │   │   └ (Business rules)
│   │   │
│   │   ├── repository.go                # Accès données
│   │   │   ├ CreatePoll()  [INSERT]
│   │   │   ├ AddOption()   [INSERT option]
│   │   │   ├ Vote()        [INSERT vote]
│   │   │   ├ HasUserVoted()[SELECT COUNT]
│   │   │   ├ GetVoteCount()[SELECT COUNT votes]
│   │   │   ├ GetPoll()     [SELECT poll]
│   │   │   ├ GetPollOptions()[SELECT options]
│   │   │   └ GetAllPolls() [SELECT all]
│   │   │
│   │   └── model.go                   # Structures de données
│   │       ├ Poll { ID, Question, Type, Options, ... }
│   │       ├ Option { ID, Text, PollID }
│   │       ├ Vote { PollID, OptionID, UserID }
│   │       └ CreatePollRequest { Question, Options, Type }
│   │
│   ├── healthcheck/
│   │   └── healthcheck.go               # Endpoint /health
│   │
│   └── test/
│       └── example_test.go              # Tests unitaires
│
├── api-node/                            # Backend Node (optionnel)
│   ├── Dockerfile
│   ├── package.json
│   ├── src/
│   │   ├── index.js         # Express server
│   │   └── db.js            # DB config
│   │
│   └── test/
│       └── example.test.js
│
├── api-gateway/                         # Reverse Proxy (Go)
│   ├── main.go                          # Gateway logic
│   │   ├ cors() middleware [headers + OPTIONS]
│   │   ├ newProxy() [httputil.ReverseProxy]
│   │   └ Path rewriting: /api/go/* → api-golang:8080/api/*
│   │
│   └── (no Dockerfile = compiled in project)
│
├── client-react/                        # Frontend React
│   ├── Dockerfile                       # Image Node.js + npm build
│   ├── package.json                     # Dépendances npm
│   ├── vite.config.js                   # Config Vite
│   ├── index.html                       # HTML root
│   ├── nginx.conf                       # Config serveur statique
│   │
│   ├── public/                          # Assets statiques
│   │
```

```
│   ├── src/
│   │   ├── main.jsx                    # Entry React
│   │   ├── App.jsx                     # Shell principal
│   │   │   ├── Navigation (Home/Polls/Create)
│   │   │   ├── QueryClientProvider
│   │   │   └── Conditional rendering
│   │   │
│   │   ├── App.css                     # Styles globaux
│   │   ├── index.css                   # Variables CSS, resets
│   │   │
│   │   ├── api.js                      # Client HTTP
│   │   │   ├── BASE = import.meta.env.VITE_GO_GATEWAY
│   │   │   ├── getUserId() [localStorage session]
│   │   │   ├── getPolls()
│   │   │   ├── getPoll(id)
│   │   │   ├── createPoll(payload)
│   │   │   ├── votePoll(id, payload)
│   │   │   └── getResults(id)
│   │   │
│   │   └── components/
│   │       ├── PollList.jsx            # Grille sondages + détails
│   │       │   ├── useState: selected, showResults, alertMessage
│   │       │   ├── useQuery: ['polls'], getPolls
│   │       │   ├── useQuery: ['poll', selected], getPoll (conditional)
│   │       │   ├── useMutation: votePoll
│   │       │   ├── Grid layout avec hover effects
│   │       │   ├── Onglets Vote/Results
│   │       │   ├── Options avec boutons vote
│   │       │   └── Alert banners (success/error)
│   │       │
│   │       ├── CreatePoll.jsx          # Formulaire création
│   │       │   ├── useState: question, options, type
│   │       │   ├── useMutation: createPoll
│   │       │   ├── Validation: min 2 options
│   │       │   ├── Textarea options (1 par ligne)
│   │       │   ├── Type selector (single/multiple)
│   │       │   └── Success/error feedback
│   │       │
│   │       ├── Results.jsx             # Visualisation votes
│   │       │   ├── useQuery: ['results', pollId], getResults
│   │       │   ├── Barre de progression par option
│   │       │   ├── Affichage pourcentages
│   │       │   ├── Total des votes
│   │       │   └── Styling gradients
│   │       │
│   │       └── __tests__/
│   │           └── CreatePoll.test.js
│   │
│   └── test/
│       └── example.test.js
│
├── docker-compose.yml                  # Orchestration services
├── Makefile                            # Commandes utiles
├── package.json                        # Dépendances root (optionnel)
└── README.md                           # Documentation utilisateur
```

5️⃣ FLUX DE DONNÉES DÉTAILLÉ

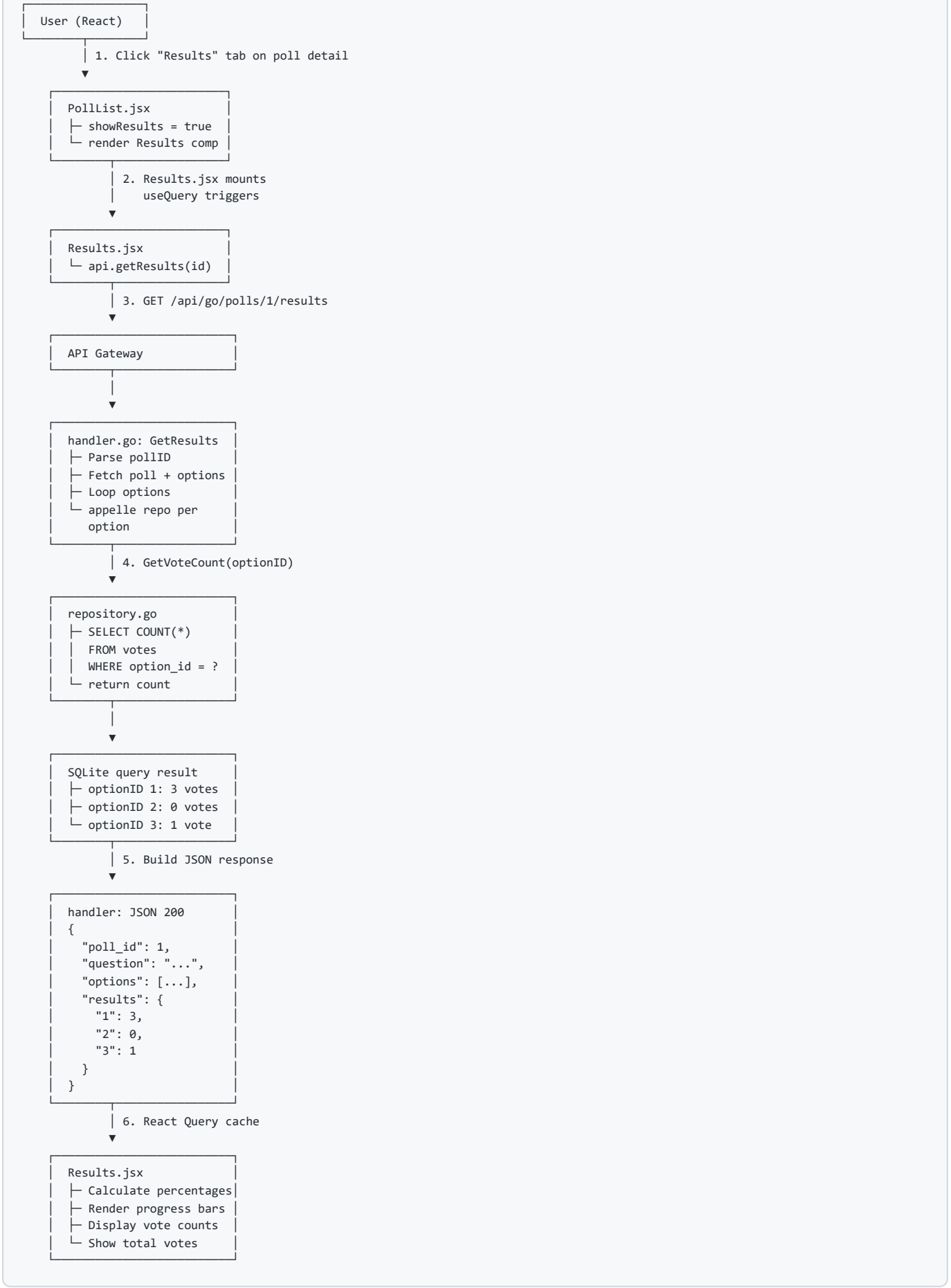
🔄 Use Case 1: Créer un Sondage



```
    │  User voit success msg   │
    │  + form clearé           │
    └──────────────────────────┘
```

🗳️ Use Case 2: Voter sur une Option

📊 Use Case 3: Voir les Résultats



6️⃣ ENDPOINTS API COMPLETS

Base URL: `http://localhost:8081/api/go`

Sondages

Méthode	Endpoint	Payload	Réponse	Code
GET	`/polls`	-	`{ value: [Poll], Count: int }`	200
GET	`/polls/:id`	-	`Poll { id, question, type, options[] }`	200
POST	`/polls`	`{ question, options[], type }`	`{ poll_id: int }`	201
GET	`/polls/:id/results`	-	`{ poll_id, question, options[], results{} }`	200
POST	`/polls/:id/vote?user_id=X`	`{ option_ids[] }`	`{ message: "Success" }`	200
POST	`/polls/:id/close`	-	`{ message: "Closed" }`	200

Schéma de Données (Payloads)

CreatePollRequest:

```
{
  "question": "Quel est le meilleur langage?",
  "options": ["Go", "Python", "JavaScript"],
  "type": "single"  // ou "multiple"
}
```

VoteRequest:

```
{
  "option_ids": [1]  // ou [1, 2, 3] pour multiple
}
```

Poll Response:

```
{
  "id": 1,
  "question": "Quel est le meilleur langage?",
  "type": "single",
  "created_by": 1,
  "created_at": "2026-02-11T14:52:11Z",
  "ends_at": null,
  "is_closed": false,
  "options": [
    { "id": 1, "poll_id": 1, "option_text": "Go" },
    { "id": 2, "poll_id": 1, "option_text": "Python" }
  ]
}
```


Results Response:


```
{
  "poll_id": 1,
  "question": "Quel est le meilleur langage?",
  "type": "single",
  "options": [...],
  "results": {
    "1": 3,
    "2": 1,
    "3": 0
  }
}
```

7️⃣ MODÈLE DE BASE DE DONNÉES

Schéma SQL

```
-- Table: polls
CREATE TABLE polls (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    question TEXT NOT NULL,
    type TEXT NOT NULL DEFAULT 'single',
    created_by INTEGER,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    ends_at DATETIME,
    is_closed BOOLEAN DEFAULT 0
);

-- Table: poll_options
CREATE TABLE poll_options (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    poll_id INTEGER NOT NULL,
    option_text TEXT NOT NULL,
    FOREIGN KEY (poll_id) REFERENCES polls(id)
);

-- Table: votes
CREATE TABLE votes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    poll_id INTEGER NOT NULL,
    option_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(poll_id, user_id),  -- Un seul vote par utilisateur par poll
    FOREIGN KEY (poll_id) REFERENCES polls(id),
    FOREIGN KEY (option_id) REFERENCES poll_options(id)
);
```

Relations

```
polls (1) ──────────────── (N) poll_options
  │
  │
  └──────────────────── (N) votes ──────────── (1) poll_options
```

Règles Métier DB

- **Unicité du vote:** `UNIQUE(poll_id, user_id)` - Un utilisateur ne peut voter qu'une fois par sondage
- **Clés étrangères:** Garantissent l'intégrité référentielle
- **Timestamps:** Enregistrement automatique de création

8️⃣ GESTION DE L'ÉTAT (Frontend)

React Query Setup

```jsx
// App.jsx
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 5000,        // 5 secondes avant refetch
      gcTime: 10 * 1000,      // 10 secondes cache retention
      retry: 1                // Retry une fois en cas d'erreur
    }
  }
})

<QueryClientProvider client={queryClient}>
  <App />
</QueryClientProvider>
```

Cache Management

Query Key	Endpoint	Comportement	Invalidation
`['polls']`	`GET /polls`	Refetch auto après 5s	On create/vote
`['poll', id]`	`GET /polls/:id`	Refetch auto après 5s	On vote
`['results', id]`	`GET /polls/:id/results`	Refetch auto après 5s	On vote

Session Management

```jsx
// localStorage
localStorage.getItem('userId')  // Récupère session user ID
localStorage.setItem('userId', randomId)  // Crée nouvelle session

// Auto-generate sur premier visit
const userId = Math.floor(Math.random() * 100000)
```

Component State

PollList.jsx:

```
const [selected, setSelected] = useState(null)      // Poll sélectionné
const [showResults, setShowResults] = useState(false) // Vue résultats?
const [hoveredId, setHoveredId] = useState(null)   // Hover effect
const [alertMessage, setAlertMessage] = useState(null) // Banner alert
const [alertType, setAlertType] = useState('success')  // success|error
```

CreatePoll.jsx:

```
const [question, setQuestion] = useState('')
const [options, setOptions] = useState(['', ''])  // Min 2 options
const [type, setType] = useState('single')
const [message, setMessage] = useState('')
```

9️⃣ STYLING & DESIGN SYSTEM

Design System (CSS Variables)

client-react/src/index.css:

```
:root {
  /* Couleurs */
  --primary: #6366f1;           /* Indigo pour boutons/accents */
  --secondary: #8b5cf6;         /* Violet pour gradients */
  --bg-dark: #0f0f0f;           /* Fond très sombre */
  --card-bg: #1a1a2e;           /* Fond cartes */
  --error-color: #ef4444;       /* Rouge erreurs */
  --success-color: #10b981;     /* Vert succès */

  /* Typographie */
  --font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', ...;
  --font-size-base: 16px;

  /* Espacements */
  --spacing-xs: 0.25rem;
  --spacing-sm: 0.5rem;
  --spacing-md: 1rem;
  --spacing-lg: 1.5rem;
  --spacing-xl: 2rem;
}
```

Animations

```
@keyframes slideDown {
  from {
    opacity: 0;
    transform: translateY(-20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@keyframes fadeIn {
  from { opacity: 0; }
  to { opacity: 1; }
}
```

Composants UI

Boutons:

- Gradient primaire sur hover
- Transition transform (translateY -2px)
- Box shadow avec couleur primaire

Cartes (Cards):

- Fond dégradé subtle
- Bordure semi-transparente
- Hover: border color + shadow enhancement

Inputs:

- Focus ring avec couleur primaire
- Background semi-transparent
- Text color contraste élevé

Alerts:

- Success: Fond vert + texte vert
- Error: Fond rouge + texte rouge
- Animation slideDown

Responsive Design

```
/* Mobile First */
@media (max-width: 768px) {
  .grid { grid-template-columns: 1fr; }  /* 1 colonne */
  .header { flex-direction: column; }
  .nav { gap: 0.5rem; }
}

/* Tablet & Desktop */
@media (min-width: 769px) {
  .grid { grid-template-columns: repeat(auto-fill, minmax(300px, 1fr)); }
}
```

🔟 FLOW UTILISATEUR COMPLET

Scénario: Nouveau Utilisateur

```
1. Ouvre http://localhost:5173
   ↓
2. Voit homepage avec 3 onglets: Home | Polls | Create Poll
   ↓
3. Click "Polls" → Voit grille de tous les sondages
   ├─ Chaque carte affiche: Question + Type + Nb options
   │
4. Click "View & Vote" sur un sondage
   ├─ Détail poll s'ouvre
   ├─ Voit les 3 options avec boutons "Vote"
   │
5. Click "Vote" sur une option
   ├─ localStorage crée userId si absent
   ├─ Envoie POST /api/go/polls/1/vote?user_id=999
   ├─ Si succès → ✅ alert green "Vote enregistré"
   ├─ Si erreur (déjà voté) → ⚠️ alert red
   ├─ Auto-refresh résultats
   │
6. Click onglet "Results"
   ├─ Voit graphique des votes
   ├─ Pour chaque option:
   │  - Barre de progression colorée
   │  - Pourcentage + nombre votes
   │
7. Click "Close" → Revient à liste
```

Flux Administrateur

```
1. Click "Create Poll"
   ↓
2. Remplir formulaire:
   - Question: "Quel langage préférez-vous?"
   - Options (textarea):
     Go
     Python
     JavaScript
   - Type: single / multiple
   ↓
3. Click "Créer"
   ├─ Validation client: min 2 options
   ├─ POST /api/go/polls
   ├─ Si succès:
   │  ├─ ✅ Alert verte
   │  ├─ Form reset
   │  ├─ Cache invalidate
   │  └─ Auto-hide message 3s
   │
4. Nouveau poll visible immédiatement dans la liste
```

1️⃣1️⃣ DÉPLOIEMENT & EXÉCUTION

Démarrage Local (Docker Compose)

```
# Démarrer tous les services
docker-compose up -d

# Services:
# - api-golang:8080      ← API sondages
# - api-node:3000        ← API utilisateurs (optionnel)
# - api-gateway:8081     ← Reverse proxy
# - client-react:5173    ← Frontend

# Arrêter
docker-compose down

# Logs
docker-compose logs -f api-golang
docker-compose logs -f client-react
```

Build Images

```
# Rebuild image Go
docker-compose up --build -d api-golang

# Rebuild image React
docker-compose up --build -d client-react

# Rebuild tous
docker-compose up --build -d
```

Variables d'Environnement

client-react:

```
VITE_GO_GATEWAY=http://localhost:8081/api/go
```

api-golang:

```
DB_PATH=/app/dev.db
PORT=8080
```

Accès

Service	URL
Frontend React	http://localhost:5173
API Gateway	http://localhost:8081
API Go Direct	http://localhost:8080/api
SQLite DB	`dev.db` (fichier)

1️⃣ 2️⃣ GESTION DES ERREURS

Frontend

```
// Erreurs réseau
if (error?.response?.status === 500) {
  showAlert("❌ Erreur serveur")
}

// Validation
if (question.length === 0) {
  showAlert("❌ Question vide")
}

// Votes déjà existants
if (error.message.includes("déjà voté")) {
  showAlert("⚠️ Vous avez déjà voté")
}
```

Backend (Handler → Service → Repository)

```
// Handler retourne erreur service
poll, err := h.Service.GetPoll(pollID)
if err != nil {
  c.JSON(http.StatusNotFound, gin.H{"error": "Poll not found"})
  return
}

// Service valide règles métier
if poll.IsClosed {
  return errors.New("poll fermé")
}

// Repository gère contraintes DB
voted, _ := r.HasUserVoted(poll.ID, userID)
if voted {
  return "utilisateur a déjà voté"
}
```

Codes HTTP

Code	Sens	Usage
200	OK	GET réussi, POST succès
201	Created	POST création poll
400	Bad Request	Validation échouée
404	Not Found	Poll inexistant
500	Server Error	Erreur DB/serveur

1️⃣3️⃣ SÉCURITÉ & BONNES PRATIQUES

Authentication/Authorization

⚠️ **Status Actuel:** Aucune auth (utilisateurs anonymes)

- Session user basée sur localStorage (random ID)
- Pas de contrôle d'accès sur les endpoints

✅ **À ajouter (Futur):**

- JWT ou sessions serveur
- Rate limiting par IP/user
- CSRF protection
- Validation stricte des inputs

Data Validation

```
// Client-side
- Min/max question length
- Min 2 options, max 10
- Type must be "single" or "multiple"

// Server-side
- Re-validate all inputs
- Type casting safe
- SQL injection prevention (parameterized queries)
```

CORS Policy

```
// api-gateway/main.go
cors.ExposeHeaders = []string{"Content-Type"}
cors.AllowOrigins = []string{"*"}  // ⚠️ À restreindre en prod
cors.AllowMethods = []string{"GET", "POST", "OPTIONS"}
```

1️⃣4️⃣ PERFORMANCE & OPTIMISATIONS

Frontend

Optimisation	Détail
Lazy Loading	React.lazy() pour routes
Code Splitting	Vite automatic + dynamic imports
Caching	React Query staleTime 5s
CSS Variables	Réduction taille CSS, theming
Images	Optimisées dans public/

Backend

Optimisation	Détail
Goroutines	Go concurrence native
Connection Pool	sqlite3 default pooling
Query Optimization	Indexes sur poll_id, user_id
Response Compression	Gzip middleware (optionnel)

Database

```
-- Index sur foreign keys
CREATE INDEX idx_votes_poll_user ON votes(poll_id, user_id);
CREATE INDEX idx_options_poll ON poll_options(poll_id);
CREATE INDEX idx_votes_option ON votes(option_id);

-- Facilite:
-- - HasUserVoted query
-- - GetVoteCount query
-- - GetPollOptions query
```

1️⃣5️⃣ TESTING

Frontend

```
# Tests React Components
npm test

# Fichiers: src/**/*.test.js ou src/**/*.spec.js
# Exemple: CreatePoll.test.js
```

Backend

```
# Tests Go
go test ./poll

# Coverage
go test -cover ./poll

# Fichiers: *_test.go
```

Testing Endpoints (cURL)

```
# Récupérer polls
curl http://localhost:8081/api/go/polls

# Créer poll
curl -X POST http://localhost:8081/api/go/polls \
  -H "Content-Type: application/json" \
  -d '{"question":"Test?","options":["Yes","No"],"type":"single"}'

# Voter
curl -X POST "http://localhost:8081/api/go/polls/1/vote?user_id=999" \
  -H "Content-Type: application/json" \
  -d '{"option_ids":[1]}'

# Résultats
curl http://localhost:8081/api/go/polls/1/results
```

1️⃣6️⃣ DÉPANNAGE COURANT

“Blank page” React

Cause: VITE_GO_GATEWAY mal configurée ou gateway down

Solution:

```
// Vérifier api.js
const BASE = import.meta.env.VITE_GO_GATEWAY || "http://localhost:8081/api/go"

// Vérifier dans docker-compose:
# environment:
#   VITE_GO_GATEWAY: http://api-gateway:8081/api/go
```

Erreur “utilisateur a déjà voté”

Cause: Même userID trouvé dans votes table

Solution:

```
// Supprimer localStorage
localStorage.removeItem('userId')

// Ou change le userID dans query param
fetch(`/api/go/polls/1/vote?user_id=${Math.random()}`)
```

Options nulles dans list

Cause: GetAllPolls() ne chargeait pas les options

Solution: ✅ Déjà corrigé - service.go GetAllPolls() maintenant charge les options

API Gateway timeout

Cause: api-golang container pas running

Solution:

```
docker-compose restart api-golang
docker-compose logs api-golang
```

1️⃣7️⃣ AMÉLIORATIONS FUTURES

Court Terme

- [ ] Pagination des sondages (actuellement tous chargés)
- [ ] Search/filter polls par question
- [ ] Export résultats (CSV, PDF)
- [ ] Polls avec deadline
- [ ] Dark/Light theme toggle

Moyen Terme

- [ ] User authentication (OAuth2/JWT)
- [ ] Permissions (propriétaire poll, modération)
- [ ] Analytics dashboard
- [ ] Real-time updates (WebSocket)
- [ ] Mobile app (React Native)

Long Terme

- [ ] Scalability (PostgreSQL, Redis cache)
- [ ] Microservices separation
- [ ] CI/CD pipeline (GitHub Actions)
- [ ] Load balancing (Nginx, Kubernetes)
- [ ] Admin panel

1️⃣8️⃣ RÉSUMÉ ARCHITECTURE

Principes Appliqués

✅ **Séparation des Responsabilités**

- Handler (HTTP) → Service (Métier) → Repository (Data)

✅ **Stateless Design**

- Chaque requête indépendante
- Session user en localStorage client

✅ **Caching Intelligent**

- React Query automatic cache + refetch
- SQLite fichier pour persistance

✅ **Error Handling**

- Try-catch côté service
- JSON error responses structurées

✅ **DRY (Don't Repeat Yourself)**

- API centralisée (api.js)
- CSS variables réutilisables
- Repository patterns

Tech Stack Summary

Couche	Tech	Raison
Présentation	React 18 + Vite	Modern, fast, ecosystem
API Gateway	Go + httputil	Lightweight, CORS/routing
Backend	Go + Gin	Compiled, concurrent, fast
Data	SQLite	Simple, file-based, reliable
Cache	React Query	Automatic, smart invalidation
Orchestration	Docker Compose	Local dev, reproducible

1️⃣9️⃣ CONCLUSION

Ce projet démontre une **architecture moderne full-stack** avec:

1. **Frontend réactif** (React + Vite) avec gestion d'état sophistiquée
2. **Backend performant** (Go + Gin) avec logique métier solide
3. **API bien structurée** avec endpoints RESTful clairs
4. **Persistance fiable** (SQLite) avec modèle de données optimal
5. **DevOps** complète (Docker, Docker Compose)
6. **UX polished** avec dark theme, animations, feedback utilisateur

Points Clés:

- Scalable (facile d'ajouter features)
- Maintenable (code organisé, bien commenté)
- Performant (Go concurrence, React Query cache)
- Sécurisé (à améliorer pour prod)
- Testable (structure clean)

Document préparé pour présentation technique *Dernière mise à jour: 11 Février 2026*