🎬 Guion pour la vidéo (10 minutes max)

1. Introduction rapide PABLO (20s)

Bonjour, nous sommes Pablo et Sara, et nous allons vous présenter notre projet de forum collaboratif développé en React côté client, et Node.js côté serveur.

2. Comparaison des diagrammes de composants PABLO (40s)

Au début du projet, nous avons réalisé un schéma conceptuel de notre architecture front-end pour le document mi-parcours, sans pratiquement coder. Maintenant que le projet est abouti, nous avons dessiné un nouveau diagramme à partir de notre code source.

Ce qui est intéressant, c'est que les deux coïncident assez bien — cela prouve que notre réflexion initiale était déjà assez réaliste et bien orientée. On avait déjà prévu la séparation claire entre Recherche, Sidebar, et les differentes pages, comme `Forum` ou `Profile`. On a aussi prévu, la hiérarchie de messages en MessageList, et également dans les replies et on a aussi un "Avatar", qu'on a appelé ProfileView qui nous donne une vision résumé d'un utilisateur

---

3. Architecture du client React SARA (1min)

Le composant principal est `PagePrincipale`, qui gère les états globaux comme : `showProfile`, `showUserList`, `showInscriptions` pour afficher la bonne page, et un autre boolean `isPrivateView` pour le mode "admin", qui active aussi un dark mode CSS

Ces états sont ensuite modifiés depuis `Sidebar`, qui est notre composant principal de navigation. On passe donc tous les setters (`setShowProfile`, `setIsPrivateView`...) comme props.

4. Interactions client via le navigateur (1min)

PABLO

Grâce à nos nombreux `console.log()`, on peut suivre très facilement ce qu'il se passe dans notre application en temps réel, via l'outil de développement du navigateur.

Par exemple, quand un utilisateur se connecte, on peut voir dans la console une confirmation côté serveur :

[LOGIN] Connexion réussie : saracas

De même, quand un message est posté, on voit apparaître immédiatement la réponse du serveur, qui contient le nouveau message ajouté :

[POST MESSAGE] Nouveau message de saracas : "Question sur les stages"

Toutes ces informations permettent de suivre l'enchaînement complet : le clic dans le composant React, l'appel de la fonction handler, la requête Axios, l'exécution côté serveur, et enfin la réponse retournée au client.

SARA

Pour les réponses aux messages, nous avons un composant appelé `ReplyList` qui s'occupe de cette logique. Il envoie une requête `axios` contenant le texte de la réponse, et utilise comme paramètre l'ID du message auquel on veut répondre.

Côté serveur, cela correspond à cette route :

// POST /api/messages/:id/reply

router.post("/messages/:id/reply", ...)

Quand cette route est appelée, on enregistre une trace dans la console :

[REPLY] saracas répond au message 1234...

Si la réponse est bien ajoutée, un autre log nous confirme :

[REPLY] Réponse ajoutée par saracas: "Merci pour ta réponse !"

Cela montre bien que notre système de log est cohérent et permet un suivi complet des actions utilisateurs, ce qui est très utile pour le développement, le débogage et même la démonstration de l'application.

6. Côté serveur – structure et services (1min)

PABLO

Notre serveur est structuré avec un fichier `api.js` qui regroupe tous les routes. On a séparé les entités (`User`, `Message`) et les services sont simples et RESTful :

Les services qu'on a ajouté à l'entité messages sont :

- Fonction save(), pour enregistrer un nouveau message dans la base de données
- Des fonctions pour la consultation des messages, comme findAllByPrivacy, qui cherche par privacité, et findWithFilters, qui cherche selon plusieurs critères. Les deux fontiones trient les par date décroissante
- Une function findAllOfUser, qui retourne tous les messages d'un utilisateur donné,

- Une function - deleteById(id), qui efface un message de la base de données

Pour la Gestion des réponses , on a quelque méthodes importantes aussi comme addReply, findRepliesByUser ou deleteReplyById

---

SARA

Pour continuer, les services qu'on a ajouté à l'entité users sont:

- La fonction create({ prenom, nom, login, email, password }), pour enregistrer un nouveau user dans la base de données, Il chiffre le mot de pase avec un hash de la bibliothèque bcrypt

- Des fonctions pour la consultation d'users, comme findByLogin, findAllFiltered, qui récupère tous les utilisateurs filtrés par leur status de validation

- Et finalement, des fonctions de changement de données, comme updateRoleById, validateById ou updateProfileById

- Pour la suppression d'un utilisateur, on a aussi une function deleteById

---

Notres différentes routes dans /api.js, elles sont
- /login, /logout, /register, /isLogged, pour les services d'autorisation
- la route /messages pour récupèrer, poster ou supprimer des messages
- et la route /users pour gérer les rôles, validations, rejects et le changement des informations d'un utilisateur

7. Zoom sur un service : POST /messages SARA (1min)

L'un des services les plus puissants que nous avons mis en place est la récupération des messages via l'endpoint : getmessages
Ce service est très **polyvalent**, car il accepte de nombreux paramètres facultatifs qui permettent de filtrer précisément les messages :

- private=true|false : pour différencier forum privé et public
- userLogin : pour filtrer par auteur
- keyword : pour chercher un mot-clé dans les titres ou contenus
- startDate, endDate : pour filtrer les messages par date
- all=true + user=ID : pour récupérer tous les messages d'un utilisateur connecté, utile dans la vue Profil

Grâce à cette flexibilité, le composant Forum peut adapter les requêtes selon les besoins de l'utilisateur : affichage général, recherche ciblée, ou historique personnel.

On va faire maintenant des preuves pour voir comment s'affiche sur le console.

Par exemple, avec le changement de forum, le filtre de recherche de privacité change. Si on filtre par login dans le fórum privé, on fait une requête plus complexe

En résumé, ce service combine **souplesse, performance et sécurité**, ce qui en fait un pilier central de notre architecture backend.

8. Gestion des sessions PABLO (30s)

Nous utilisons express-session pour garder les informations de l'utilisateur connecté dans req.session.user.

On y stocke :

- l'_id, utile pour les requêtes internes sans avoir à passer de paramètres,
- le role, utilisé souvent dans notre API pour gérer les droits d'accès,
- et des infos immuables comme le login et l'email, affichées dans la Sidebar.

La session crée une cookie à la connexion, détruite au logout.

Cette cookie permet aussi de rediriger l'utilisateur au démarrage de l'app selon s'il est connecté ou non.

Grâce à cela, le backend peut travailler directement avec les données de session sans passer par le body ou les params.

9. Fonctionnalité dont on est fiers SARA (40s)

Ce qu'on aime particulièrement et on a ajouté comme bonus, c'est l'équilibre entre l'espace public et privé du forum — l'admin peut basculer d'un mode à l'autre, voir les inscriptions en attente, valider ou refuser, et ça se reflète visuellement dans l'interface avec le dark mode.

10. Bonus SARA (30s)

Nous avons aussi ajouté :

- Un **compteur dynamique** des inscriptions en attente, basé sur un état React et une requête MongoDB avec countDocuments(), qui se met à jour en temps réel pour les administrateurs.

Avec plus de temps, nous aimerions améliorer la recherche côté API pour permettre de **filtrer les utilisateurs** directement dans les fenêtres "Découvrir personnes" et "Inscriptions en attente", de la même manière que dans le forum

11. Conclusion personnelle PABLO (30s)

Ce projet nous a beaucoup appris, autant sur React que sur Node.js.
Il reste encore beaucoup à faire, mais ce que nous avons construit pose une base solide.

Nous espérons pouvoir continuer à l'améliorer, aussi bien côté front-end qu'au niveau du backend — et que cette expérience nous serve pour aller plus loin dans nos futurs projets.