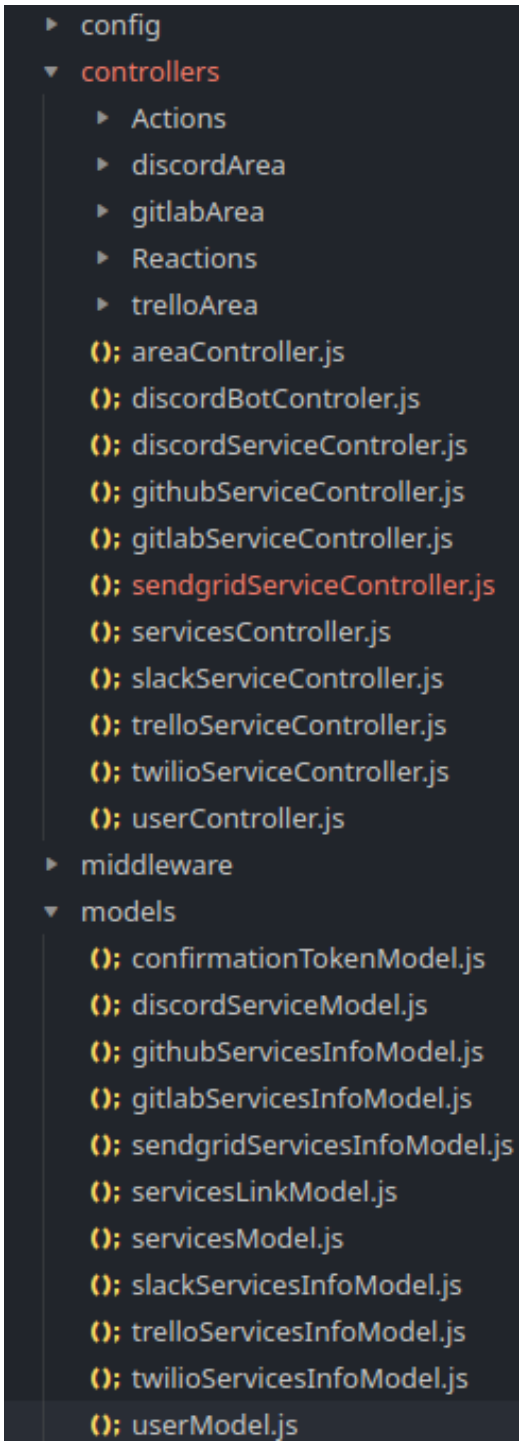
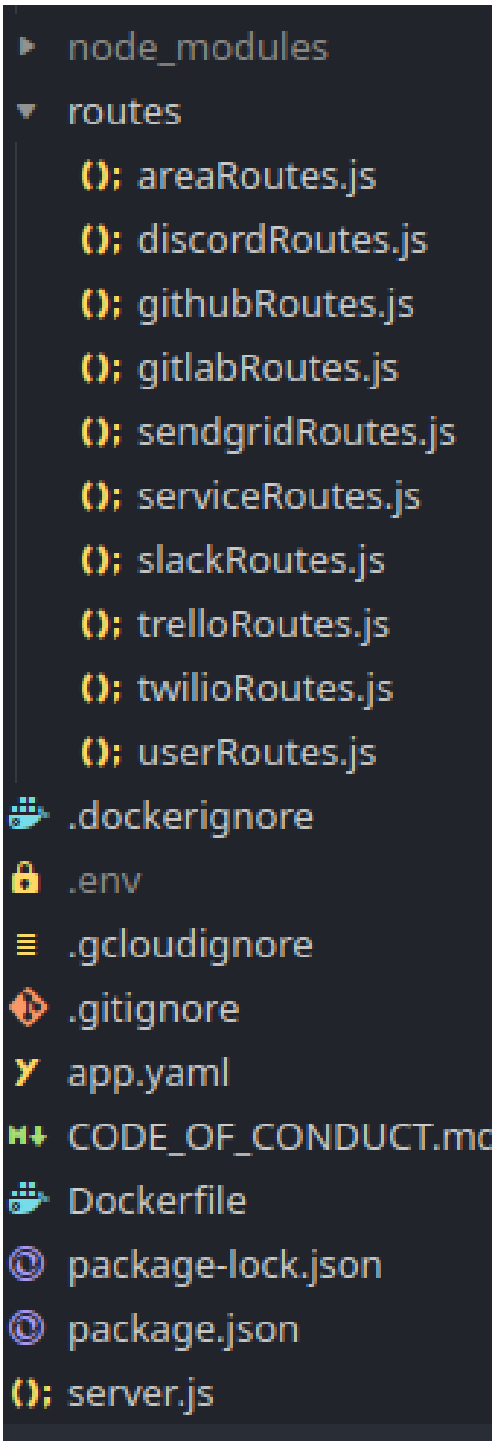
DOCUMENTATION TECHNIQUE BACK-END

Area

Coulibaly Yannis, Fontaine Melvyn,
Aflalo Gary, Corentin Calvo Wang
Zhiwen, Zhou Tony

ARCHITECTURE





ARCHITECTURE

↳ MODELS/

Ce dossier contient tous les modèles utilisés dans l'application. Les modèles sont principalement des schema Mongoose.

ARCHITECTURE

↳ Models/

↳ servicesModel.js

```
const mongoose = require('mongoose');

const servicesSchema = new mongoose.Schema({
    nameServices: {
      type: String,
      required: true,
    },
    userIdRef: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
    },
    accessToken: {
      type: String,
      required: true,
    },
    refreshToken: {
        type: String,
        required: true,
    },
}, { timestamps: true });

module.exports = mongoose.model('services', servicesSche
```

Ce fichier permet de créer le schema de la table "Services" sur la base de donnée Mongoose

Il contient 4 éléments (nameServices, userIdRef, accessToken et refreshToken.

ARCHITECTURE
↳ Models/
↳ serviceLinkModel.js

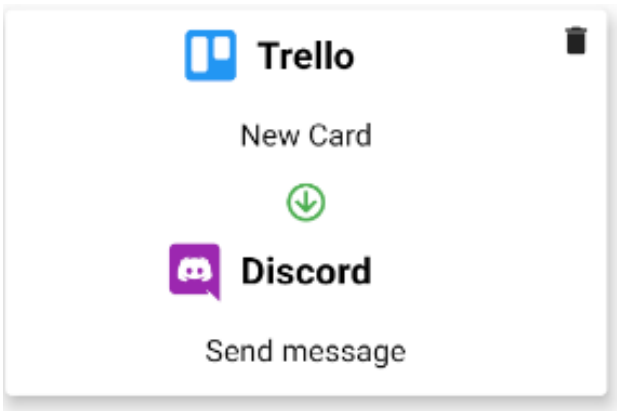
```
const mongoose = require('mongoose');

const servicesLinkSchema = new mongoose.Schema({
    Services: {
      type: Array(Object),
      required: true,
    },
    userIdRef: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
    },
    title: {
      type: String,
      required: false,
    },
}, { timestamps: true });
```

Ce modele permet de savoir quelle action / reaction est lié à quel service.

- Services est de type Array , il contient 2 éléments (le nom du service de l'action et le nom du service de la réaction

- userIdRef est l'id de l'utilisateur voulant faire le lien entre 2 services

-Title : Nom donnée à l'action/réaction



ARCHITECTURE

↳ Models/

↳ trelloServicesInfoModel.js

```
const mongoose = require('mongoose');

const trelloServiceInfoSchema = new mongoose.Schema({
    serviceIdRef: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
    },
    tokenType : {
        type: String,
        required: true,
    },
    description: {
      type: String,
      required: false,
    },
    trelloBoardName: {
        type: String,
        required: false,
    },
    trelloBoardID: {
        type: String,
        required: false,
    },
    trelloListName: {
        type: String,
        required: false,
    },
    trelloListID: {
        type: String,
        required: false,
    },
    trelloUserID: {
        type: String,
        required: false,
    },
}, { timestamps: true });
```

Ce model explique ce que nous sauvegardons en base de données lorque un utilisateur crée le service Trello

ARCHITECTURE

↳ CONTROLLERS/

Ce dossier contient tout les controllers sur les services que peux utiliser un utilisateur

ARCHITECTURE

↳ CONTROLLERS/

↳ GITHUB/

Ce dossier contient toutes les fonctions que peux utiliser un utilisateur pour faire des actions ou réactions

ARCHITECTURE
↳ CONTROLLERS/
↳ GITHUBSERVICECONTROLLERS/

Exemple de fonction dans le fichier

```
exports.getUserRepo = async (req, res) => {
    const { userId } = req.body;

    try {
        const github = await servicesModel.findOne({ userIdRef: userId, nameServices: "GitHub"});
        if (!github) {
            return res.status(404).json({ error: 'Failed to get user\'s GitHub info from DB' });
        }
        const {accessToken} = github;
        const githubRes = await getGithubApiRep('get', '/user/repos', accessToken, {});
        if (!githubRes) {
            return res.status(403).json({ error: 'Try to get GitHub user info from API failed'});
        }
        res.status(200).json({ success: githubRes });
    } catch (error) {
        console.log(error);
        res.status(500).json({ error: 'An error has occurred' });
    }
}
```

Fonction getUserRepo qui permet de récuperer les projet d'un utilisateur sur Github. Elle neccessite d'envoyer le userId dans le body de la fonction

ARCHITECTURE
↳ ROUTES/

Ce dossier contient toutes les routes de notre API REST

ARCHITECTURE

↳ ROUTES

↳ GITLABROUTES.JS

Ce fichier contient toutes les routes concernant le service gitlab

```
module.exports = function (router) {
    router.get('/gitlab/getRepos', isAuth, (req, res) => {
        gitlabController.getReposUser(req, res);
    });
    router.post('/gitlab/setService', [
        check('projectId', 'projectId cannot be empty').notEmpty(),
        check('projectName', 'projectName cannot be empty').notEmpty(),
    ], isAuth, (req, res) => {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).send({ error: errors.errors[0].msg });
        }
        gitlabController.setGitlabService(req, res);
    });
    router.all('/gitlab/hook', (req, res) => {
        gitlabController.gitlabHook(req, res);
    });
};
```

ARCHITECTURE

↳ routes

↳ trelloRoutes.js/

Ce fichier contient les routes lié au service Trello

ARCHITECTURE

↳ Pages/

↳ userRoutes.js/

Cette page contient toutes les routes lié à l'utilisateur
Ex: Login , Register ...

ARCHITECTURE
↳ CONFIG

Ce dossier contient la liste des Actions/Reactions ainsi que les variables d'environnement

ARCHITECTURE

↳ config

↳ areaGithubDictionnary.js

Ce fichier contient un dictionnaire des actions/reaction du service Gitlab

```
exports.GitHubActionDico = new Map([
...
    ["New push", "push"],
    ["New issue", "issues"],
    ["Create branch", "create"],
    ["Delete branch", "delete"],
    ["Create repo", "repository"],
]);
```

Tout les autres fichiers du dossier config ayant comme suffixe "Dictionnary" dispose d'un dictionnaire ayant le même rôle que celui présenté ci-dessus

ARCHITECTURE

↳ config

↳ config.js

Ce fichier contient toutes les variables d'environnement du projet

```
module.exports = ({
  mongoURI: process.env.MONGO_URI,
  portBack: process.env.PORT,
  secretToken: process.env.TOKEN_SECRET,
  sendgridKey:process.env.SENDGRID_API_KEY,
  urlConfirm:process.env.WEB_URL,
  TrelloApiKey : process.env.TRELLO_API_KEY,
  TrelloHookURI : process.env.TRELLO_REDIRECT_HOOK_URI,
  GITLAB_HOOK_URL : process.env.GITLAB_HOOK_URL,
  GithubHookURI : process.env.GITHUB_REDIRECT_HOOK_URI,
  DiscordApiKey : process.env.DISCORD_CLIENT_ID,
  DiscordBotToken : process.env.DISCORD_BOT_TOKEN,
  SlackClientId: process.env.SLACK_CLIENT_ID,
  SlackClientSecret: process.env.SLACK_CLIENT_SECRET,
  SlackSigningSecret: process.env.SLACK_SIGNING_SECRET,
  SlackBotToken: process.env.SLACK_BOT_TOKEN,
  TwillioToken: process.env.TWILIO_TOKEN,
  TwillioSid: process.env.TWILIO_SID,
  senderSendGrid: process.env.SENDGRID_EMAIL_SENDER,
});
```