Documentation technique

Ticket Bus CM

UV PROJET CODE Team

3 juillet 2025

Table des matières

Introduction

Cette documentation technique a été générée automatiquement à partir des commentaires JSDoc du projet Node.js/Express Ticket Bus CM. Elle présente l'ensemble des modules, routes, modèles et utilitaires du backend.

0.0.1 Modules

app

Configure les middlewares, les routes, la gestion des erreurs et les options CORS pour lAPI Ticket Bus CM.

controllers/adminController

Permet dobtenir des statistiques, de gérer les entités principales et de communiquer avec les utilisateurs. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

controllers/authController

authController.js

controllers/busController

Permet dajouter, modifier et lister les bus. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

controllers/paymentController

Gère linitiation, la réception de webhook et la génération de tickets. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

controllers/reservationController

Permet de créer une réservation, de consulter les réservations de lutilisateur connecté et de récupérer les sièges réservés pour un horaire donné. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

controllers/terminalController

Permet dajouter, modifier et lister les terminaux. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

controllers/ticketController

Permet de récupérer le ticket associé à une réservation et la liste des sièges réservés pour un horaire donné. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

controllers/userController

Permet de récupérer, supprimer et répondre aux messages. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

routes/admin

Toutes les routes sont protégées par le middleware admin. Permet la gestion complète des entités du système Ticket Bus CM.

routes/auth

Permet la gestion de linscription, de la connexion, de la demande et vérification dOTP, et de la modification du profil utilisateur.

routes/bus

Permet dajouter, modifier et lister les bus via lAPI REST.

routes/payment

Permet dinitier un paiement et de gérer les webhooks NotchPay via lAPI REST.

routes/reservation

Permet de créer une réservation et de consulter les réservations de lutilisateur connecté via lAPI REST.

routes/schedule

Permet de créer, lister les horaires et récupérer les sièges réservés via lAPI REST.

routes/ticket

Permet de récupérer un ticket par lID de la réservation via lAPI REST.

routes/user

Permet à lutilisateur de consulter, supprimer et répondre à des messages via lAPI REST. Toutes les routes sont protégées par le middleware dauthentification.

0.0.2 Functions

connectDB() Promise.<void>

Initialise la connexion à la base de données MongoDB à partir de lURI stockée dans la variable denvironnement MONGODB_URI. Arrête le serveur en cas déchec de connexion.

connectDB()

Initialise la connexion à la base de données MongoDB avec Mongoose. Utilise lURI stockée dans la variable denvironnement MONGODB_URI. Arrête le serveur en cas déchec de connexion.

createAdminUser() Promise.<void>

À lancer manuellement pour initialiser un compte admin. Utilise Mongoose pour la connexion et Bcrypt pour le hash du mot de passe.

sendOtpMail(email, otp) Promise.<void>

Utilise Nodemailer pour envoyer un email avec un code OTP à lutilisateur. Les identifiants sont récupérés depuis les variables denvironnement.

utils/sendOtpMail(email, subject, htmlContent) Promise.<void>

Envoie un email générique (OTP ou message personnalisé)

0.0.3 Typedefs

Bus : Object

Bus.js

EmailVerification : Object

EmailVerification.js

Message : Object

Message.js

Payment : Object

Payment.js

Reservation : Object

Reservation.js

Schedule : Object

Schedule.js

Terminal : Object

Terminal.js

Ticket : Object

Ticket.js

User : Object

User.js

0.0.4 app

Configure les middlewares, les routes, la gestion des erreurs et les options CORS pour lAPI Ticket Bus CM.

Brief : Fichier principal de configuration de lapplication Express pour Ticket Bus CM.
Routes : - GET / : Test API - /api/auth : Authentification - /api/bus : Gestion des bus - /api/terminaux : Gestion des terminaux - /api/schedules : Gestion des horaires - /api/reservations : Gestion des réservations - /api/payments : Paiements - /api/tickets : Tickets - /api/admin : Administration - /api/user : Utilisateur
Date : 2024-06-01
See

— server.js pour le démarrage du serveur
— config/db.js pour la connexion à la base de données

Version : 1.0
Author : UV PROJET CODE Team

0.0.5 controllers/adminController

Permet dobtenir des statistiques, de gérer les entités principales et de communiquer avec les utilisateurs. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur administrateur pour la gestion centralisée des bus, terminaux, horaires, réservations, paiements, tickets, utilisateurs et messages.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

— controllers/adminController
  — *static*
    — .getDashboardStats(req, res) Promise.<void>
    — .getAllBuses(req, res) Promise.<void>
    — .createBus(req, res) Promise.<void>
  — *inner*
    — ~updateBus(id) Object
    — ~deleteBus(id) Object
    — ~getAllTerminals() Array
    — ~createTerminal(nom, ville, adresse) Object
    — ~updateTerminal(id) Object
    — ~deleteTerminal(id) Object
    — ~getAllSchedules() Array
    — ~createSchedule(bus, terminal_depart, terminal_arrivee, date_depart, date_arrivee, prix, places_disponibles) Object
    — ~updateSchedule(id) Object
    — ~deleteSchedule(id) Object
    — ~getAllReservations() Array
    — ~updateReservationStatus(id, statut) Object
    — ~getAllPayments() Array
    — ~getAllTickets() Array
    — ~getAllUsers() Array
    — ~updateUser(id, nom, email, telephone, type) Object
    — ~deleteUser(id) Object
    — ~sendMessage(to, subject, body) Object
    — ~getAllMessages() Array

— ~getUserMessages(userId) → Array
— ~markAsRead(id, userId) → Object
— ~getInbox() → Array

controllers/adminController.getDashboardStats(req, res) → Promise.<void>

Récupère les statistiques du dashboard admin (bus, terminaux, réservations, revenus, etc.).
Kind : static method of controllers/adminController
Returns : Promise.<void> - Réponse JSON contenant les statistiques et réservations récentes ou un message derreur.
Throws :
— 500 Si une erreur survient lors de la récupération.
See
— module :models/Bus
— module :models/Terminal
— module :models/Schedule
— module :models/Reservation
— module :models/Ticket
— module :models/Payment
— module :models/User

Param	Type	Description
req	Express.Request	Requête HTTP Express.
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/admin/dashboard', getDashboardStats);
```

controllers/adminController.getAllBuses(req, res) → Promise.<void>

Liste tous les bus.
Kind : static method of controllers/adminController
Returns : Promise.<void> - Réponse JSON contenant la liste des bus ou un message derreur.
Throws :
— 500 Si une erreur survient lors de la récupération.
See : module :models/Bus

Param	Type	Description
req	Express.Request	Requête HTTP Express.
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/admin/buses', getAllBuses);
```

controllers/adminController.createBus(req, res) → Promise.<void>

Crée un nouveau bus.
Kind : static method of controllers/adminController
Returns : Promise.<void> - Réponse JSON contenant le bus créé ou un message derreur.
Throws :

— 500 Si une erreur survient lors de la création.
See : module :models/Bus

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : numero, capacite, compagnie, type_bus).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/admin/buses', createBus);
```

controllers/adminController~updateBus(id) Object

Modifie un bus existant.
Kind : inner method of controllers/adminController
Returns : Object - Bus modifié ou message derreur
Route : PUT /api/admin/buses/ :id

Param	Type	Description
id	string	Identifiant du bus

controllers/adminController~deleteBus(id) Object

Supprime un bus.
Kind : inner method of controllers/adminController
Returns : Object - Message de succès ou derreur
Route : DELETE /api/admin/buses/ :id

Param	Type	Description
id	string	Identifiant du bus

controllers/adminController~getAllTerminals() Array

Liste tous les terminaux.
Kind : inner method of controllers/adminController
Returns : Array - Liste des terminaux
Route : GET /api/admin/terminals

controllers/adminController~createTerminal(nom, ville, adresse) Object

Crée un nouveau terminal.
Kind : inner method of controllers/adminController
Returns : Object - Terminal créé
Route : POST /api/admin/terminals

Param	Type	Description
nom	string	Nom du terminal
ville	string	Ville du terminal
adresse	string	Adresse du terminal

controllers/adminController~updateTerminal(id) Object

Modifie un terminal existant.
Kind : inner method of controllers/adminController
Returns : Object - Terminal modifié ou message derreur
Route : PUT /api/admin/terminals/ :id

Param	Type	Description
id	string	Identifiant du terminal

controllers/adminController~deleteTerminal(id) Object

Supprime un terminal.
Kind : inner method of controllers/adminController
Returns : Object - Message de succès ou derreur
Route : DELETE /api/admin/terminals/ :id

Param	Type	Description
id	string	Identifiant du terminal

controllers/adminController~getAllSchedules() Array

Liste tous les horaires.
Kind : inner method of controllers/adminController
Returns : Array - Liste des horaires
Route : GET /api/admin/schedules

controllers/adminController~createSchedule(bus, terminal_depart, terminal_arrivee, date_depart, date_arrivee, prix, places_disponibles) Object

Crée un nouvel horaire.
Kind : inner method of controllers/adminController
Returns : Object - Horaire créé
Route : POST /api/admin/schedules

Param	Type	Description
bus	string	Bus associé
terminal_depart	string	Terminal de départ
terminal_arrivee	string	Terminal darrivée
date_depart	string	Date de départ
date_arrivee	string	Date darrivée
prix	number	Prix du trajet
places_disponibles	number	Nombre de places disponibles

controllers/adminController~updateSchedule(id) Object

Modifie un horaire existant.
Kind : inner method of controllers/adminController
Returns : Object - Horaire modifié ou message derreur
Route : PUT /api/admin/schedules/ :id

Param	Type	Description
id	string	Identifiant de lhoraire

controllers/adminController~deleteSchedule(id) Object

Supprime un horaire.
Kind : inner method of controllers/adminController
Returns : Object - Message de succès ou derreur
Route : DELETE /api/admin/schedules/ :id

Param	Type	Description
id	string	Identifiant de lhoraire

controllers/adminController~getAllReservations() Array

Liste toutes les réservations.
Kind : inner method of controllers/adminController
Returns : Array - Liste des réservations
Route : GET /api/admin/reservations

controllers/adminController~updateReservationStatus(id, statut) Object

Modifie le statut dune réservation.
Kind : inner method of controllers/adminController
Returns : Object - Réservation modifiée ou message derreur
Route : PUT /api/admin/reservations/ :id

Param	Type	Description
id	string	Identifiant de la réservation
statut	string	Nouveau statut

controllers/adminController~getAllPayments() Array

Liste tous les paiements confirmés.
Kind : inner method of controllers/adminController
Returns : Array - Liste des paiements
Route : GET /api/admin/payments

controllers/adminController~getAllTickets() Array

Liste tous les tickets.
Kind : inner method of controllers/adminController
Returns : Array - Liste des tickets
Route : GET /api/admin/tickets

controllers/adminController~getAllUsers() Array

Liste tous les utilisateurs (clients).
Kind : inner method of controllers/adminController
Returns : Array - Liste des utilisateurs
Route : GET /api/admin/users

controllers/adminController~updateUser(id, nom, email, telephone, type) Object

Modifie un utilisateur.
Kind : inner method of controllers/adminController
Returns : Object - Utilisateur modifié ou message derreur
Route : PUT /api/admin/users/ :id

Param	Type	Description
id	string	Identifiant de lutilisateur
nom	string	Nom (optionnel)
email	string	Email (optionnel)
telephone	string	Téléphone (optionnel)
type	string	Type dutilisateur (optionnel)

controllers/adminController~deleteUser(id) Object

Supprime un utilisateur (sauf soi-même).
Kind : inner method of controllers/adminController
Returns : Object - Message de succès ou derreur
Route : DELETE /api/admin/users/ :id

Param	Type	Description
id	string	Identifiant de lutilisateur

controllers/adminController~sendMessage(to, subject, body) Object

Envoie un message à un ou plusieurs utilisateurs.
Kind : inner method of controllers/adminController
Returns : Object - Message envoyé
Route : POST /api/admin/messages

Param	Type	Description
to	Array.<string>	Destinataires
subject	string	Sujet du message
body	string	Contenu du message

controllers/adminController~getAllMessages() Array

Liste tous les messages envoyés.
Kind : inner method of controllers/adminController
Returns : Array - Liste des messages
Route : GET /api/admin/messages

controllers/adminController~getUserMessages(userId) Array

Liste les messages reçus par un utilisateur donné.
Kind : inner method of controllers/adminController
Returns : Array - Liste des messages
Route : GET /api/admin/messages/user/ :userId

Param	Type	Description
userId	string	Identifiant de lutilisateur

controllers/adminController~markAsRead(id, userId) Object

Marque un message comme lu pour un utilisateur.
Kind : inner method of controllers/adminController
Returns : Object - Message de confirmation
Route : POST /api/admin/messages/ :id/read

Param	Type	Description
id	string	Identifiant du message
userId	string	Identifiant de lutilisateur

controllers/adminController~getInbox() Array

Récupère la boîte de réception de ladmin (messages reçus).
Kind : inner method of controllers/adminController
Returns : Array - Liste des messages reçus
Route : GET /api/admin/inbox

0.0.6 controllers/authController

authController.js
Brief : Contrôleur dauthentification pour la gestion des utilisateurs : inscription, connexion, OTP, et mise à jour du profil. Gère aussi la vérification demail via OTP.

- controllers/authController
  - .register(req, res) Promise.<void>
  - .login(req, res) Promise.<void>
  - .updateProfile(nom, email, telephone, photo) Object
  - .requestSignupOtp(email) Object
  - .verifyOtpAndRegister(email, otp, nom, mot_de_passe, telephone) Object
  - .requestResetOtp(email) Object
  - .resetPassword(email, otp, newPassword) Object

controllers/authController.register(req, res) Promise.<void>

Inscrit un nouvel utilisateur.

Kind : static method of controllers/authController

Returns : Promise.<void> - Réponse JSON avec message de succès ou derreur.

Throws :

— 400 Si lemail est déjà utilisé.

— 500 Si une erreur survient lors de linscription.

See : module :models/User

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : nom, email, mot_de_passe, telephone, type).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/auth/register', register);
```

controllers/authController.login(req, res) Promise.<void>

Connecte un utilisateur existant.

Kind : static method of controllers/authController

Returns : Promise.<void> - Réponse JSON avec token JWT et infos utilisateur ou message derreur.

Throws :

— 400 Si email ou mot de passe incorrect.

— 500 Si une erreur survient lors de la connexion.

See : module :models/User

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : email, mot_de_passe).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/auth/login', login);
```

controllers/authController.updateProfile(nom, email, telephone, photo) Object

Met à jour le profil de lutilisateur connecté.

Kind : static method of controllers/authController

Returns : Object - Utilisateur mis à jour ou message derreur

Route : PUT /api/auth/profile

Param	Type	Description
nom	string	Nouveau nom (optionnel)
email	string	Nouvel email (optionnel)
telephone	string	Nouveau téléphone (optionnel)
photo	string	Nouvelle photo (optionnel)

controllers/authController.requestSignupOtp(email) Object

Demande lenvoi dun OTP pour linscription.
Kind : static method of controllers/authController
Returns : Object - Message de succès ou derreur
Route : POST /api/auth/request-otp

Param	Type	Description
email	string	Email pour lequel envoyer lOTP

controllers/authController.verifyOtpAndRegister(email, otp, nom, mot_de_passe, telephone) Object

Vérifie lOTP et crée le compte utilisateur.
Kind : static method of controllers/authController
Returns : Object - Message de succès ou derreur
Route : POST /api/auth/verify-otp

Param	Type	Description
email	string	Email à vérifier
otp	string	Code OTP reçu
nom	string	Nom de lutilisateur
mot_de_passe	string	Mot de passe
telephone	string	Numéro de téléphone

controllers/authController.requestResetOtp(email) Object

Demande lenvoi dun OTP pour la réinitialisation du mot de passe.
Kind : static method of controllers/authController
Returns : Object - Message de succès ou derreur
Route : POST /api/auth/request-reset-otp

Param	Type	Description
email	string	Email pour lequel envoyer lOTP

controllers/authController.resetPassword(email, otp, newPassword) Object

Réinitialise le mot de passe après vérification de lOTP.
Kind : static method of controllers/authController
Returns : Object - Message de succès ou derreur
Route : POST /api/auth/reset-password

Param	Type	Description
email	string	Email à vérifier
otp	string	Code OTP reçu
newPassword	string	Nouveau mot de passe

0.0.7 controllers/busController

Permet dajouter, modifier et lister les bus. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur pour la gestion des bus : ajout, modification et listing.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

— controllers/busController
  — .ajouterBus(req, res) Promise.<void>
  — .modifierBus(req, res) Promise.<void>
  — .listerBus(req, res) Promise.<void>

controllers/busController.ajouterBus(req, res) Promise.<void>

Ajoute un nouveau bus.

Kind : static method of controllers/busController
Returns : Promise.<void> - Réponse JSON avec message de succès ou derreur.
Throws :

— 400 Si un champ obligatoire est manquant.
— 500 Si une erreur survient lors de lajout.

See : module :models/Bus

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : numero, capacite, compagnie, type_bus, status).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/bus', ajouterBus);
```

controllers/busController.modifierBus(req, res) Promise.<void>

Modifie un bus existant.

Kind : static method of controllers/busController
Returns : Promise.<void> - Réponse JSON avec message de succès ou derreur.
Throws :

— 404 Si le bus nest pas trouvé.
— 500 Si une erreur survient lors de la modification.

See : module :models/Bus

Param	Type	Description
req	Express.Request	Requête HTTP Express (params : id, body : champs à modifier).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.put('/bus/ :id', modifierBus);
```

controllers/busController.listerBus(req, res) Promise.<void>

Liste tous les bus.

Kind : static method of controllers/busController

Returns : Promise.<void> - Réponse JSON contenant la liste des bus ou un message derreur.

Throws :

— 500 Si une erreur survient lors de la récupération.

See : module :models/Bus

Param	Type	Description
req	Express.Request	Requête HTTP Express.
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/bus', listerBus);
```

0.0.8 controllers/paymentController

Gère linitiation, la réception de webhook et la génération de tickets. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur pour la gestion des paiements et de lintégration NotchPay.

Date : 2024-06-01

Version : 1.0

Author : UV PROJET CODE Team

— controllers/paymentController

— ~~.processPayment(req, res) Promise.<void>~~

— .handleNotchPayWebhook(req, res) Promise.<void>

— .initiatePayment(req, res) void

~~controllers/paymentController.processPayment(req, res) Promise.<void>~~

Déprécié

(Obsolète) Simule le paiement et la génération de ticket.

Kind : static method of controllers/paymentController

Returns : Promise.<void> - Réponse JSON derreur (cette route nest plus utilisée).

Param	Type	Description
req	Express.Request	Requête HTTP Express.

Param	Type	Description
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/payment/process', processPayment);
```

controllers/paymentController.handleNotchPayWebhook(req, res) Promise.<void>

Gère le webhook NotchPay pour la confirmation de paiement et la génération du ticket.

Kind : static method of controllers/paymentController

Returns : Promise.<void> - Réponse JSON selon le traitement du paiement.

Throws :

- 400 Si la référence de marchand est manquante.
- 404 Si la réservation nest pas trouvée.
- 500 Si une erreur survient lors du traitement du paiement.

See

- module :models/Payment
- module :models/Reservation
- module :models/Ticket

Param	Type	Description
req	Express.Request	Requête HTTP Express (webhook NotchPay).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/payment/webhook', handleNotchPay-
Webhook);
```

controllers/paymentController.initiatePayment(req, res) void

Kind : static method of controllers/paymentController

Brief : Initialise un paiement NotchPay pour une réservation donnée.

Param	Type	Description
req	Object	Requête HTTP Express (body : reservationId).
res	Object	Réponse HTTP Express.

Example

```
[] initiatePayment(req, res);
```

0.0.9 controllers/reservationController

Permet de créer une réservation, de consulter les réservations de lutilisateur connecté et de récupérer les sièges réservés pour un horaire donné. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur pour la gestion des réservations : création, consultation et sièges réservés.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

— controllers/reservationController
  — .creerReservation(req, res) Promise.<void>
  — .mesReservations(req, res) Promise.<void>
  — .getReservedSeats(req, res) Promise.<void>

controllers/reservationController.creerReservation(req, res) Promise.<void>

Crée une nouvelle réservation et génère un ticket avec QR code.

Kind : static method of controllers/reservationController
Returns : Promise.<void> - Réponse JSON contenant la réservation et le ticket généré, ou un message derreur.
Throws :

— 404 Si lhoraire nest pas trouvé.
— 400 Si pas assez de places disponibles.
— 500 Si une erreur survient lors de la création.

See

— module :models/Reservation
— module :models/Ticket
— module :models/Schedule
— module :models/User
— module :models/Message
— module :utils/sendOtpMail

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : schedule, nombre_places, seat). Lutilisateur doit être authentifié (req.user.userId).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/reservation', creerReservation);
```

controllers/reservationController.mesReservations(req, res) Promise.<void>

Récupère les réservations de lutilisateur connecté.

Kind : static method of controllers/reservationController
Returns : Promise.<void> - Réponse JSON contenant la liste des réservations ou un message derreur.
Throws :

— 500 Si une erreur survient lors de la récupération.

See : module :models/Reservation

Param	Type	Description
req	Express.Request	Requête HTTP Express (utilisateur authentifié).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/mes-reservations', mesReservations);
```

controllers/reservationController.getReservedSeats(req, res) Promise.<void>

Récupère la liste des sièges réservés pour un horaire donné.

Kind : static method of controllers/reservationController

Returns : Promise.<void> - Réponse JSON contenant la liste des sièges réservés ou un message derreur.

Throws :

— 500 Si une erreur survient lors de la récupération.

See : module :models/Ticket

Param	Type	Description
req	Express.Request	Requête HTTP Express (params : scheduleId).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/reserved-seats/ :scheduleId', getReserved-
Seats);
```

0.0.10 controllers/scheduleController

Permet de créer, lister et récupérer les horaires de bus. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur pour la gestion des horaires : création, listing et récupération par ID.

Date : 2024-06-01

Version : 1.0

Author : UV PROJET CODE Team

— controllers/scheduleController
  — .creerHoraire(req, res) Promise.<void>
  — .listerHoraires(req, res) Promise.<void>
  — .recupererHoraire(req, res) void

controllers/scheduleController.creerHoraire(req, res) Promise.<void>

Crée un nouvel horaire pour un bus donné.

Kind : static method of controllers/scheduleController

Returns : Promise.<void> - Réponse JSON avec lhoraire créé ou un message derreur.

Throws :

— 404 Si le bus nest pas trouvé.
— 400 Si lorigine et la destination sont identiques.
— 500 Si une erreur survient lors de la création.

See
— module :models/Schedule
— module :models/Bus

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : bus_id, origine, destination, heure_depart, heure_arrivee, prix).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/horaires', creerHoraire);
```

controllers/scheduleController.listerHoraires(req, res) Promise.<void>

Liste tous les horaires, avec possibilité de filtrer par origine, destination et date.

Kind : static method of controllers/scheduleController

Returns : Promise.<void> - Réponse JSON contenant la liste des horaires ou un message derreur.

Throws :
— 500 Si une erreur survient lors de la récupération.

See : module :models/Schedule

Param	Type	Description
req	Express.Request	Requête HTTP Express (query : origine, destination, date).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/horaires', listerHoraires);
```

controllers/scheduleController.recupererHoraire(req, res) void

Kind : static method of controllers/scheduleController

Brief : Récupère un horaire par son identifiant.

Param	Type	Description
req	Object	Requête HTTP Express (params : id).
res	Object	Réponse HTTP Express.

Example

```
[] recupererHoraire(req, res);
```

0.0.11 controllers/terminalController

Permet dajouter, modifier et lister les terminaux. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur pour la gestion des terminaux : ajout, modification et listing.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

— controllers/terminalController
  — .ajouterTerminal(req, res) Promise.<void>
  — .modifierTerminal(req, res) Promise.<void>
  — .listerTerminaux(req, res) Promise.<void>

controllers/terminalController.ajouterTerminal(req, res) Promise.<void>

Ajoute un nouveau terminal.

Kind : static method of controllers/terminalController
Returns : Promise.<void> - Réponse JSON avec message de succès ou derreur.
Throws :

— 500 Si une erreur survient lors de lajout.

See : module :models/Terminal

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : ville_destination, terminal_info).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.post('/terminaux', ajouterTerminal);
```

controllers/terminalController.modifierTerminal(req, res) Promise.<void>

Modifie un terminal existant.

Kind : static method of controllers/terminalController
Returns : Promise.<void> - Réponse JSON avec message de succès ou derreur.
Throws :

— 404 Si le terminal nest pas trouvé.
— 500 Si une erreur survient lors de la modification.

See : module :models/Terminal

Param	Type	Description
req	Express.Request	Requête HTTP Express (params : id, body : champs à modifier).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.put('/terminaux/ :id', modifierTerminal);
```

controllers/terminalController.listerTerminaux(req, res) Promise.<void>

Liste tous les terminaux.

Kind : static method of controllers/terminalController

Returns : Promise.<void> - Réponse JSON contenant la liste des terminaux ou un message derreur.

Throws :

— 500 Si une erreur survient lors de la récupération.

See : module :models/Terminal

Param	Type	Description
req	Express.Request	Requête HTTP Express.
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/terminaux', listerTerminaux);
```

0.0.12 controllers/ticketController

Permet de récupérer le ticket associé à une réservation et la liste des sièges réservés pour un horaire donné. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur pour la gestion des tickets : récupération par réservation et sièges réservés.

Date : 2024-06-01

Version : 1.0

Author : UV PROJET CODE Team

— controllers/ticketController
  — .getTicketByReservationId(req, res) Promise.<void>
  — .getReservedSeats(req, res) Promise.<void>

controllers/ticketController.getTicketByReservationId(req, res) Promise.<void>

Récupère le ticket associé à une réservation pour lutilisateur connecté.

Kind : static method of controllers/ticketController

Returns : Promise.<void> - Réponse JSON contenant le ticket ou un message derreur.

Throws :

— 404 Si la réservation nest pas trouvée ou non autorisée.
— 404 Si le ticket nest pas trouvé.
— 500 Si une erreur survient lors de la récupération.

See

— module :models/Ticket
— module :models/Reservation

Param	Type	Description
req	Express.Request	Requête HTTP Express (params : reservationId, utilisateur authentifié).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/ticket/ :reservationId', getTicketByReservationId);
```

controllers/ticketController.getReservedSeats(req, res) Promise.<void>

Récupère la liste des sièges réservés pour un horaire donné.

Kind : static method of controllers/ticketController

Returns : Promise.<void> - Réponse JSON contenant la liste des sièges réservés ou un message derreur.

Throws :

— 500 Si une erreur survient lors de la récupération.

See : module :models/Ticket

Param	Type	Description
req	Express.Request	Requête HTTP Express (params : scheduleId).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/reserved-seats/ :scheduleId', getReservedSeats);
```

0.0.13 controllers/userController

Permet de récupérer, supprimer et répondre aux messages. Toutes les fonctions sont asynchrones et renvoient des réponses JSON.

Brief : Contrôleur utilisateur pour la gestion des messages et des interactions avec ladmin.

Date : 2024-06-01

Version : 1.0

Author : UV PROJET CODE Team

— controllers/userController
  — .getMyMessages(req, res) Promise.<void>
  — .deleteMyMessage(req, res) Promise.<void>
  — .replyToAdmin(req, res) Promise.<void>

controllers/userController.getMyMessages(req, res) Promise.<void>

Récupère les messages de lutilisateur connecté.

Kind : static method of controllers/userController

Returns : Promise.<void> - Réponse JSON contenant la liste des messages reçus ou un message derreur.

Throws :

— 500 Si une erreur survient lors de la récupération.

See : module :models/Message

Param	Type	Description
req	Express.Request	Requête HTTP Express (utilisateur authentifié).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.get('/user/messages', getMyMessages);
```

controllers/userController.deleteMyMessage(req, res) Promise.<void>

Permet à lutilisateur de supprimer un de ses messages.
Kind : static method of controllers/userController
Returns : Promise.<void> - Réponse JSON de succès ou message derreur.
Throws :
— 500 Si une erreur survient lors de la suppression.
See : module :models/Message

Param	Type	Description
req	Express.Request	Requête HTTP Express (params : id, utilisateur authentifié).
res	Express.Response	Réponse HTTP Express.

Example

```
[] // Appel depuis une route Express router.delete('/user/messages/ :id', deleteMyMessage);
```

controllers/userController.replyToAdmin(req, res) Promise.<void>

Permet à lutilisateur de répondre à ladmin.
Kind : static method of controllers/userController
Returns : Promise.<void> - Réponse JSON de succès ou message derreur.
Throws :
— 500 Si une erreur survient lors de lenvoi du message.
See
— module :models/Message
— module :models/User

Param	Type	Description
req	Express.Request	Requête HTTP Express (body : subject, body, utilisateur authentifié).
res	Express.Response	Réponse HTTP Express.

Example

[] // Appel depuis une route Express router.post('/user/reply-admin', replyToAdmin) ;

0.0.14 routes/admin

Toutes les routes sont protégées par le middleware admin. Permet la gestion complète des entités du système Ticket Bus CM.

Brief : Routes dadministration pour la gestion centralisée des bus, terminaux, horaires, réservations, paiements, tickets, utilisateurs et messages.
Date : 2024-06-01
Route : GET /api/admin/dashboard Statistiques du dashboard
Route : GET/POST/PUT/DELETE /api/admin/buses Gestion des bus
Route : GET/POST/PUT/DELETE /api/admin/terminals Gestion des terminaux
Route : GET/POST/PUT/DELETE /api/admin/schedules Gestion des horaires
Route : GET/PUT /api/admin/reservations Gestion des réservations
Route : GET /api/admin/payments Gestion des paiements
Route : GET /api/admin/tickets Gestion des tickets
Route : GET/PUT/DELETE /api/admin/users Gestion des utilisateurs
Route : POST/GET/PUT /api/admin/messages Gestion des messages
Route : GET /api/admin/messages/inbox Boîte de réception admin
Version : 1.0
Author : UV PROJET CODE Team

0.0.15 routes/auth

Permet la gestion de linscription, de la connexion, de la demande et vérification dOTP, et de la modification du profil utilisateur.

Brief : Routes dauthentification : connexion, inscription, OTP, modification de profil.
Date : 2024-06-01
Route : POST /api/auth/login Connexion utilisateur
Route : PUT /api/auth/profile Modification du profil utilisateur (protégée)
Route : POST /api/auth/request-signup-otp Demande dOTP pour inscription
Route : POST /api/auth/verify-otp-register Vérification OTP et inscription
Version : 1.0
Author : UV PROJET CODE Team

0.0.16 routes/bus

Permet dajouter, modifier et lister les bus via lAPI REST.

Brief : Routes pour la gestion des bus (ajout, modification, listing).
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

0.0.17 routes/payment

Permet dinitier un paiement et de gérer les webhooks NotchPay via lAPI REST.

Brief : Routes pour linitiation des paiements et la gestion des webhooks de paiement.
Date : 2024-06-01
Route : POST /api/payment/initiate-payment Initie un paiement (protégée)

Route : POST /api/payment/notchpay-webhook Webhook NotchPay (non protégée)
Version : 1.0
Author : UV PROJET CODE Team

0.0.18 routes/reservation

Permet de créer une réservation et de consulter les réservations de lutilisateur connecté via lAPI REST.

Brief : Routes pour la création et la consultation des réservations de tickets.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

0.0.19 routes/schedule

Permet de créer, lister les horaires et récupérer les sièges réservés via lAPI REST.

Brief : Routes pour la gestion des horaires de bus et la récupération des sièges réservés.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

0.0.20 routes/ticket

Permet de récupérer un ticket par lID de la réservation via lAPI REST.

Brief : Routes pour la gestion et la récupération des tickets de réservation.
Date : 2024-06-01
Route : GET /api/ticket/ :reservationId Récupère un ticket par lID de la réservation (protégée)
Version : 1.0
Author : UV PROJET CODE Team

0.0.21 routes/user

Permet à lutilisateur de consulter, supprimer et répondre à des messages via lAPI REST. Toutes les routes sont protégées par le middleware dauthentification.

Brief : Routes utilisateur pour la gestion des messages et la communication avec ladmin.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

0.0.22 connectDB() Promise.<void>

Initialise la connexion à la base de données MongoDB à partir de lURI stockée dans la variable denvironnement MONGODB_URI. Arrête le serveur en cas déchec de connexion.

Kind : global function
Returns : Promise.<void> - Une promesse qui se résout si la connexion est réussie
Throws :

— Error En cas déchec de connexion

Brief : Configuration de la connexion à la base de données MongoDB avec Mongoose.
Date : 2024-06-01
See : https ://mongoosejs.com/docs/
Version : 1.0
Author : UV PROJET CODE Team

0.0.23 connectDB()

Initialise la connexion à la base de données MongoDB avec Mongoose. Utilise lURI stockée dans la variable denvironnement MONGODB_URI. Arrête le serveur en cas déchec de connexion.

Kind : global function

0.0.24 createAdminUser() Promise.<void>

À lancer manuellement pour initialiser un compte admin. Utilise Mongoose pour la connexion et Bcrypt pour le hash du mot de passe.

Kind : global function
Returns : Promise.<void> - Une promesse qui se résout quand lutilisateur admin est créé
Throws :

— Error En cas déchec de connexion ou de création

Brief : Script utilitaire pour créer un utilisateur administrateur par défaut dans la base de données.
Date : 2024-06-01
See : models/User.js pour le modèle utilisateur
Version : 1.0
Author : UV PROJET CODE Team
Example

```
[] // Depuis la ligne de commande : node createAdmin.js
```

0.0.25 sendOtpMail(email, otp) Promise.<void>

Utilise Nodemailer pour envoyer un email avec un code OTP à lutilisateur. Les identifiants sont récupérés depuis les variables denvironnement.

Kind : global function
Returns : Promise.<void> - Une promesse qui se résout quand lemail est envoyé
Throws :

— Error En cas déchec denvoi de lemail

Brief : Utilitaire pour envoyer un email contenant un code OTP à lutilisateur pour la vérification demail.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team

Param	Type	Description
email	string	Adresse email du destinataire
otp	string	Code OTP à envoyer

Example

```
[] await sendOtpMail('user@mail.com', '123456');
```

0.0.26 utils/sendOtpMail(email, subject, htmlContent) Promise.<void>

Envoie un email générique (OTP ou message personnalisé)
Kind : global function
Returns : Promise.<void> - Une promesse qui se résout quand lemail est envoyé
Throws :
— Error En cas déchec denvoi de lemail

Param	Type	Description
email	string	Destinataire
subject	string	Sujet de lemail
htmlContent	string	Contenu HTML du message

Example

```
[] await sendMail('user@mail.com', 'Sujet', '<b>Votre code : 123456</b>');
```

0.0.27 Bus : Object

Bus.js
Kind : global typedef
Brief : Modèle Mongoose pour les bus.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
numero	String	Numéro unique du bus
capacite	Number	Capacité du bus
compagnie	String	Compagnie du bus
type_bus	String	Type de bus (Standard, VIP, Climatisé)
status	String	Statut du bus (actif, inactif)
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const bus = new Bus({ numero : 'BUS001', capacite : 50, compagnie : 'Express', type_bus :
'VIP' });
```

0.0.28 EmailVerification : Object

EmailVerification.js

Kind : global typedef
Brief : Modèle Mongoose pour la vérification demail par OTP.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
email	String	Email à vérifier
otp	String	Code OTP envoyé
expiresAt	Date	Date dexpiration de lOTP
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const verif = new EmailVerification({ email : 'test@mail.com', otp : '123456', expiresAt : new Date() });
```

0.0.29 Message : Object

Message.js
Kind : global typedef
Brief : Modèle Mongoose pour les messages entre utilisateurs et admins.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
to	Array.<ObjectId>	Destinataires du message
from	ObjectId	Expéditeur du message
subject	String	Sujet du message
body	String	Contenu du message
sentAt	Date	Date denvoi
readBy	Array.<ObjectId>	Utilisateurs ayant lu le message
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const msg = new Message({ to : [userId], from : adminId, subject : 'Info', body : 'Bienvenue !' });
```

0.0.30 Payment : Object

Payment.js
Kind : global typedef
Brief : Modèle Mongoose pour les paiements liés aux réservations.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
reservation_id	ObjectId	Référence vers la réservation
montant	Number	Montant payé
moyen	String	Moyen de paiement (Mobile Money, NotchPay, Test)
status	String	Statut du paiement (en attente, succès, échec)
transaction_id	String	Identifiant de la transaction
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const payment = new Payment({ reservation_id, montant : 10000, moyen : 'NotchPay', status : 'succès' });
```

0.0.31 Reservation : Object

Reservation.js
Kind : global typedef
Brief : Modèle Mongoose pour les réservations de tickets.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
user	ObjectId	Référence vers lutilisateur
schedule	ObjectId	Référence vers lhoraire
nombre_places	Number	Nombre de places réservées
statut	String	Statut de la réservation (en_attente, confirmée, annulée, terminée)
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const reservation = new Reservation({ user, schedule, nombre_places : 2 });
```

0.0.32 Schedule : Object

Schedule.js
Kind : global typedef
Brief : Modèle Mongoose pour les horaires de bus.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
bus	ObjectId	Référence vers le bus
terminal_depart	ObjectId	Terminal de départ
terminal_arrivee	ObjectId	Terminal darrivée
date_depart	Date	Date et heure de départ
date_arrivee	Date	Date et heure darrivée
prix	Number	Prix du trajet
places_disponibles	Number	Nombre de places restantes
status	String	Statut de lhoraire (programmé, parti, arrivé, annulé)
seat_map	Array	Carte des sièges
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const schedule = new Schedule({ bus, terminal_depart, terminal_arrivee, date_depart :
new Date(), prix : 5000 });
```

0.0.33 Terminal : Object

Terminal.js
Kind : global typedef
Brief : Modèle Mongoose pour les terminaux de départ/arrivée.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
nom	String	Nom du terminal
ville	String	Ville du terminal
adresse	String	Adresse du terminal
createdAt	Date	Date de création (auto-gérée par Mongoose)

Name	Type	Description
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const terminal = new Terminal({ nom : 'Gare Yaoundé', ville : 'Yaoundé', adresse : 'Centre-ville' }) ;
```

0.0.34 Ticket : Object

Ticket.js
Kind : global typedef
Brief : Modèle Mongoose pour les tickets générés après réservation et paiement.
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
reservation_id	ObjectId	Référence vers la réservation
user_id	ObjectId	Référence vers lutilisateur
schedule	ObjectId	Référence vers lhoraire
qr_code	String	Données du QR code
seat	String	Siège assigné
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const ticket = new Ticket({ reservation_id, user_id, schedule, seat : 'A1' }) ;
```

0.0.35 User : Object

User.js
Kind : global typedef
Brief : Modèle Mongoose pour les utilisateurs (clients et admins).
Date : 2024-06-01
Version : 1.0
Author : UV PROJET CODE Team
Properties

Name	Type	Description
nom	String	Nom de lutilisateur
email	String	Email unique
mot_de_passe	String	Mot de passe hashé

Name	Type	Description
type	String	Type dutilisateur (client, admin)
telephone	String	Numéro de téléphone
photo	String	Photo de profil (optionnelle)
createdAt	Date	Date de création (auto-gérée par Mongoose)
updatedAt	Date	Date de dernière modification (auto-gérée par Mongoose)

Example

```
[] const user = new User({ nom : 'Jean', email : 'jean@mail.com', mot_de_passe : '...', type : 'client', telephone : '123456789' }) ;
```