Laravel API for Financial Management

This is a Laravel-based API application for managing financial operations, including users, accounts (comptes), transactions, and notifications. It provides endpoints for creating and managing clients, admins, accounts, and transactions, with features like role-based access and logging.

Features

- User management (clients and admins)
- Account (Compte) management with different types and currencies
- Transaction processing
- Notification services (e.g., via Twilio)
- Role-based middleware for access control
- API documentation via Swagger

Deployment to Render

This application is configured for deployment on Render, a cloud platform for hosting web applications. The `render.yaml` file defines the services, including a PHP web service and a PostgreSQL database.

Prerequisites

- A GitHub repository containing your project code.
- A Render account.

Deployment Steps

1. **Connect Your Repository to Render:**
    - Log in to your Render dashboard.
    - Go to the "Web Services" section and click "New Web Service".
    - Select "Build and deploy from a Git repository" and connect your GitHub repository.
    - Choose the repository and branch (e.g., `main`).
2. **Configure the Service:**
    - Render will automatically detect the `render.yaml` file and use it to set up the services.
    - The configuration includes:
        - **Web Service:** Runs the Laravel application using PHP.
        - **Database Service:** PostgreSQL database for data storage.
    - Review and confirm the settings. The build command and start command are predefined in `render.yaml`.
3. **Set Environment Variables:**
    - In the Render dashboard, go to your web service's settings and add the following environment variables under "Environment".

- Some variables are pre-set in `render.yaml`, but you may need to add or override others based on your requirements.
- Required environment variables (based on `.env.example` and configuration):

Variable	Description	Example Value	Required
`APP_NAME`	Application name	`Laravel API`	Yes
`APP_ENV`	Environment (set to `production`)	`production`	Yes (set in render.yaml)
`APP_DEBUG`	Debug mode (disable in production)	`false`	Yes (set in render.yaml)
`APP_KEY`	Application key (auto-generated)	(auto-generated)	Yes (set in render.yaml)
`APP_URL`	Base URL of the application	`https://your-app.onrender.com`	Yes
`DB_CONNECTION`	Database connection type	`pgsql`	Yes (set in render.yaml)
`DATABASE_URL`	Database connection string	(auto-provided by Render)	Yes (set in render.yaml)
`LOG_CHANNEL`	Logging channel	`stack`	Yes (set in render.yaml)
`LOG_LEVEL`	Log level	`error`	Yes (set in render.yaml)
`CACHE_DRIVER`	Cache driver	`file`	Yes (set in render.yaml)
`QUEUE_CONNECTION`	Queue connection	`sync`	Yes (set in render.yaml)
`SESSION_DRIVER`	Session driver	`file`	Yes (set in render.yaml)
`MAIL_MAILER`	Mail driver	`smtp`	Optional
`MAIL_HOST`	SMTP host	`smtp.example.com`	Optional
`MAIL_PORT`	SMTP port	`587`	Optional

PROF

Variable	Description	Example Value	Required
`MAIL_USERNAME`	SMTP username	`your-email@example.com`	Optional
`MAIL_PASSWORD`	SMTP password	`your-password`	Optional
`MAIL_FROM_ADDRESS`	From email address	`hello@example.com`	Optional
`MAIL_FROM_NAME`	From name	`${APP_NAME}`	Optional
`AWS_ACCESS_KEY_ID`	AWS access key (for S3, etc.)	`your-key`	Optional
`AWS_SECRET_ACCESS_KEY`	AWS secret key	`your-secret`	Optional
`AWS_DEFAULT_REGION`	AWS region	`us-east-1`	Optional
`AWS_BUCKET`	S3 bucket name	`your-bucket`	Optional
`PUSHER_APP_ID`	Pusher app ID (for broadcasting)	`your-id`	Optional
`PUSHER_APP_KEY`	Pusher key	`your-key`	Optional
`PUSHER_APP_SECRET`	Pusher secret	`your-secret`	Optional
`PUSHER_APP_CLUSTER`	Pusher cluster	`mt1`	Optional

- **Note:** Variables marked as "set in render.yaml" are automatically configured. For others, add them in the Render dashboard if needed for your setup (e.g., email, AWS, Pusher).
- If using notifications (e.g., Twilio), add relevant variables like `TWILIO_ACCOUNT_SID`, `TWILIO_AUTH_TOKEN`, `TWILIO_PHONE_NUMBER` if implemented.

PROF

4. **Deploy:**
    - Click "Create Web Service". Render will build and deploy your application.
    - The build process includes:
        - Installing Composer dependencies.
        - Generating the application key.
        - Caching configuration, routes, and views.
        - Running database migrations.
    - Once deployed, your API will be available at the provided URL (e.g., `https://your-app.onrender.com`).
5. **Access the API:**
    - The API endpoints are available at the root URL.

- For API documentation, visit `/api/documentation` if Swagger is configured (check `config/l5-swagger.php`).
- Use tools like Postman or curl to interact with endpoints such as `/api/users`, `/api/comptes`, `/api/transactions`.

Database Setup

- The database is automatically provisioned as a PostgreSQL service in `render.yaml`.
- Migrations are run during the build process, so your database schema will be set up automatically.
- No manual database setup is required.

Additional Configurations

- **Notifications:** If using Twilio or other services, ensure the corresponding environment variables are set and the services are configured in your code.
- **File Storage:** If using AWS S3 for file uploads, configure the AWS variables accordingly.
- **Caching and Queues:** Currently set to file and sync drivers, suitable for small-scale deployments. For production, consider Redis or other drivers.
- **Security:** Ensure `APP_DEBUG` is `false` and use HTTPS in production.

Troubleshooting

- **Build Failures:** Check the build logs in Render for errors related to dependencies or migrations.
- **Database Issues:** Verify that migrations ran successfully and the `DATABASE_URL` is correctly set.
- **Environment Variables:** Double-check that all required variables are added in the Render dashboard.
- **API Access:** Confirm the service is running and accessible. Check logs for any runtime errors.

For more details on Render deployment, refer to the Render Documentation.

Local Development

To run the application locally:

PROF

1. Clone the repository.
2. Copy `.env.example` to `.env` and update the variables (e.g., database settings).
3. Run `composer install`.
4. Generate key: `php artisan key:generate`.
5. Set up a local database (e.g., MySQL or PostgreSQL) and update `.env`.
6. Run migrations: `php artisan migrate`.
7. Start the server: `php artisan serve`.

Contributing

Contributions are welcome! Please follow standard Laravel practices.

License

This project is licensed under the MIT License.