Welcome to CoreConnect

CoreConnect is RB2's framework for developing platforms that performant, robust and easy to develop with. CoreConnect enables developers and business analysts to develop and grow customer platforms.

It consists of:

- A ready-made **.NET Core backend** that offers a 'standardised' *set of GraphQL queries* to interact with a composable landscape
- A full suite of **connectors and plugins** that can be integrated into the backend, including a *CTO portal* to monitor landscape performance
- A ready-made front-end **NextJS commerce application**, setup with *authentication*, *cookies*, *localisation* and *Storyblok integration*, including a large component library

Architecture

High level architecture

CoreConnect sets up a ready made backend architecture for your composable landscape. This backend exposes a set of standardised

GraphQL queries to any front-end application.

Backend

The backend sets up a `.NET Core` application and is split into two projects: `Gateway` and `Functions`:

`Gateway` project

The gateway is responsible for handling **synchronous** requests. For example, `Commerce Gateway` is responsible for returning products and filters for displaying in a PLP, or PDP. But also when adding or removing items from cart, `Gateway` will interact directly with the commerce engine to perform these actions.

`Functions` project

The `Functions` project is responsible for handling **asynchronous** requests. The functions project can be setup with *Azure Service Bus* or *RabbitMQ*. `Functions` integrates external systems and provides the event-driven nature of the platform. Think of it like syncing systems in the background, sending transactional emails via an ESP when orders have been paid or creating an account.

Use cases where `Functions` will be would responsible for:

1. Retrieve products and their details from a PIM and store these in the commerce engine
2. Retrieve prices and stock levels for products from the ERP and update the products in the commerce engine
3. Send a transactional email via an ESP when the customer creates an account
4. Send a transactional email via an ESP when an order has been paid

Frontend

The front-end is built on `NextJS` and has a couple of key features already set up. Things like authentication, cookies and a full-fledged component library. A full overview of the structure of the boilerplate can be found under directory structure.

Directory structure

While setting up a new project, the CLI setup 3 folders:

Folder	Content
backend	containing a boilerplate backend (`gateway` and `functions` projects), with plugins (setup during initi installation process) already pre-configured in `appsettings.json`
frontend	containing a boilerplate NextJS application and component library for a new project
infrastructure	

`/backend` directory

Inside the `/backend` directory, you'll find two main projects: `Gateway` and `Functions`. Each project serves a specific purpose in the backend architecture.

Gateway Project

The `Gateway` project acts as the central hub for handling requests, managing errors, and orchestrating communication with external services. Here’s an overview of the subdirectories within the `Gateway` project:

- **`/ErrorFilters`**: - The `ErrorFilters` directory contains filters responsible for handling and filtering errors in the application.
- **`/ErrorHandlers`**: - In the `ErrorHandlers` directory, you can find modules dedicated to handling various types of errors gracefully.
- **`/GraphQL`**: - The `GraphQL` directory hosts GraphQL-related files, including schemas and resolvers.
- **`/Health`**: - Within the `Health` directory, you’ll find components related to the health check functionality of your application.
- **`/Messages`**: - The `Messages` directory contains modules responsible for managing and processing messages within the application.
- **`/Properties`**: - The `Properties` directory may hold additional configuration files or resources specific to the Gateway project.
- `/Program.cs`: The entry point for the Gateway project.
- `/Mutation.cs`: Contains GraphQL mutation-related logic.
- `<ProjectName>.Gateway.CommerceTools.csproj`: Project file for the CommerceTools integration.
- `<ProjectName>.Gateway.Scayle.csproj`: Project file for the Scayle integration.
- `appsettings.json`: Configuration file for the Gateway project.

Functions Project

The `Functions` project is designed for handling background tasks, consuming messages, and executing specific functions. Here’s an overview of the subdirectories within the `Functions` project:

- **`/bin`**: The `bin` directory may contain compiled binaries or other output files generated during the build process.
- **`/Consumers`**: The `Consumers` directory hosts modules responsible for consuming messages from message queues or similar systems.
- **`/obj`**: The `obj` directory typically contains temporary build-related files.
- **`/Properties`**: Similar to the `Properties` directory in the `Gateway` project, this directory may hold additional configuration files or resources specific to the Functions project.
- **`/Settings`**: The `Settings` directory contains configuration files or modules related to the specific settings of your Functions.
- `appsettings.json`: Configuration file for the Functions project.
- `Program.cs`: The entry point for the Functions project.
- `<ProjectName>.Functions.csproj`: Project file for the Functions project.

`/frontend` directory

Welcome to the documentation for the directory structure of the front-end of your Next.js project. This guide provides an overview of the organization within the `/frontend` directory, offering insights into each subdirectory's purpose and content.

Inside the `/frontend` directory, you will discover the `<ProjectName>` directory, which is the root of your Next.js project. This directory is structured as follows:

- **`/app`**: The `app` directory houses the main application files, including pages, layouts, and other core components.
- **`/components`**: In the `components` directory, you'll find reusable React components used across various pages and sections of your application.
- **`/fragments`**: The `fragments` directory contains smaller, reusable components or fragments that can be used in multiple places.
- **`/helpers`**: The `helpers` directory stores utility functions and helper modules used throughout the application.
- **`/hooks`**: In the `hooks` directory, you'll find custom React hooks that encapsulate and abstract complex logic for reusability.
- **`/libraries`**: The `libraries` directory is reserved for third-party libraries or external scripts used in your project.
- **`/mock`**: The `mock` directory is dedicated to mock data and services for testing and development purposes.
- **`/queries`**: In the `queries` directory, you can store GraphQL or other data fetching queries used by your components.
- **`/services`**: The `services` directory contains modules responsible for interacting with external services, APIs, or backend systems.
- **`/state`**: The `state` directory holds files related to state management.
- **`/tests`**: The `tests` directory is where you can store your unit tests, integration tests, or any testing-related files.
- **`/types`**: The `types` directory contains TypeScript type definitions used across the application.
- `.env`: Environment variables configuration.
- `middleware.ts`: Custom middleware for your application.
- `next-env.d.ts`: TypeScript declaration file for Next.js environment.
- `next.config.js`: Next.js configuration file.
- `package.json`: Node.js package configuration.
- `playwright.config.ts`: Configuration file for Playwright testing.
- `postcss.config.js`: Configuration for PostCSS, a CSS preprocessor.

- `tailwind.config.js`: Configuration for Tailwind CSS, a utility-first CSS framework.
- `tsconfig.json`: TypeScript configuration file.

`/infrastructure` directory

The `infrastructure` folder in your Composable Commerce platform contains essential Pulumi files responsible for deploying and managing your infrastructure on Azure. This folder is structured to streamline the deployment process and ensure the scalability and reliability of your platform.

- **`index.ts`**: Entry point file for the Pulumi project, orchestrating the deployment process.
- **`README.md`**: Documentation providing essential information about the structure and usage of the infrastructure folder.
- **`tsconfig.json`**: Configuration file for TypeScript compiler settings.
- **`setupResourceGroup.ts`**: File to set up the Azure Resource Group, providing a logical container for Azure resources.
- **`setupFunctions.ts`**: File for configuring serverless Azure Functions.
- **`setupGateway.ts`**: File for setting up the gateway, possibly managing API Gateway or similar services.
- **`setupKeyVault.ts`**: File for configuring Azure Key Vault, which securely stores and manages sensitive information.
- **`setupFE.ts`**: File for configuring front-end resources.
- **`setupContainerRegistry.ts`**: File for setting up Azure Container Registry to store and manage container images.
- **`Pulumi.yaml`**: Configuration file specifying project metadata and dependencies.
- **`package.json`**: Node.js package configuration file listing project dependencies.
- **`pnpm-lock.yaml`**: Lock file for ensuring consistent installations across environments.
- **`Pulumi.dev.yaml`**: Development-specific configuration file for Pulumi.
- **`/azure-service-bus`**: Directory possibly containing configuration for Azure Service Bus.

Features overview

CoreConnect tries to strike a balance between **short term quickstart** and **long term stability and flexibility**.

With that in mind, we try to provide a solid ‘structure’ on which to build long-term platforms. To speed projects up, we also include a set of features and components for rapid developing functionalities.

Priority therefor lies with 1) a solid architecture, 2) solid ‘universal’ set of endpoints, 3) solid ‘universal’ component library, then specific features. The use case is currently oriented around *e-commerce*.

Supported features

E-commerce features

- Homepage
- Landing pages
- Blog
- Internationalization
- Corporate content pages
- Different product types
- Free text search
- Faceted filtering
- Categories
- Product listing pages
- Product detail pages
- Full fledged shopping cart
- Wishlist
- Product variant support
- Image gallery
- Advanced price calculation
- Vouchers / Coupons
- Multiple payment methods

CoreConnect can use multiple plugins and adapters to connect to third party tools and applications.

Tool Name	Purpose
Azure	Cloud computing platform by Microsoft
Commercetools	E-commerce platform for building online stores
Docker	Containerization platform for building, shipping, and running applications
Mollie	Payment gateway for online transactions
OpenTelemetry	Observability framework for collecting telemetry data
Pay	Payment processing service

Tool Name	Purpose
RabbitMQ	Message broker for handling asynchronous messaging
Scayle	Headless commerce decouples backend and frontend, allowing for faster innovations and effective scaling. Accelerate your time-to-market speed and create tailored experiences for any touchpoint.
SendGrid	Email delivery and marketing platform
SQLite	Embedded relational database management system
Storyblok	Headless CMS (Content Management System)
Application Insights	Application performance monitoring and analytics platform
CTO	Chief Technology Officer (CTO) - likely a role, not a tool
Feed Generator	Tool for generating data feeds
Timer	Tool for timing events
Adyen	Payment processing platform for online and in-store payments
AWS	Cloud computing platform by Amazon Web Services
Buckaroo	Payment service provider for online payments
Cloudflare	Web infrastructure and website security company
Contentful	Headless Content Management System (CMS)
DigitalOcean	Cloud infrastructure provider for developers
Dynamics	Suite of enterprise resource planning (ERP) software
Fly.io	Platform for deplying and hosting full-stack applications.
GCP	Google Cloud Platform, offering cloud computing services
Ingenico	Payment processing solutions provider
Netsuite	Cloud-based business management software suite
Vercel	Platform for deploying and hosting web applications
Tweakwise	E-commerce personalization and optimization platform
Akeneo	Product information management (PIM) software
Zuora	Subscription billing and management platform