# Contribution Guidelines

## Commit messages

Please use [conventional commits](https://www.conventionalcommits.org) (also known as \_semantic commits\_) to ensure consistent and descriptive commit messages when submitting PRs.

## General guidelines

Every example should follow a number of guidelines.

### Example guidelines

- Simple and minimal instead of complex and production-ready

- Focus on one specific Prisma use case

- Easily extensible

- Require global installation of the Prisma 2 CLI but all other dependencies (e.g. `graphqlgen`) only as dev dependencies and invoked via a script (e.g. `npm run graphqlgen`)

### README guidelines

- Clear instructions that end with a running example

- Include instructions how to \_use\_ the running example

## Example structure

For each language, there should be one example that shows how to:

- use Prisma client in a \*\*script\*\*

- build a \*\*GraphQL server\*\* using Prisma client

- build a \*\*GraphQL server\*\* using Prisma client \*\*with authentication and permissions\*\*

- build a \*\*GraphQL server\*\* using Prisma client \*\*with realtime GraphQL subscriptions\*\*

- build a \*\*REST API\*\* with Prisma client

- build a \*\*CLI\*\* application with Prisma client

The schemas and developed APIs should be consistent across languages. This enables easier testing of the examples. The domain for all applications (except the CLI app) is a simple blogging application.

<!--

Here's an overview of the used datamodels and APIs:

### Script

<details><summary>View datamodel</summary>

`datamodel.prisma`:

```graphql

type User {

id: ID! @id

email: String! @unique

name: String

posts: [Post!]!

}

type Post {

id: ID! @id

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean! @default(value: "false")

title: String!

content: String

author: User!

}

```

</details>

### GraphQL server

<details><summary>View datamodel and GraphQL schema</summary>

`datamodel.prisma`:

```graphql

type User {

id: ID! @id

email: String! @unique

name: String

posts: [Post!]!

}

type Post {

id: ID! @id

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean! @default(value: "false")

title: String!

content: String

author: User!

}

```

`schema.graphql`:

```graphql

scalar DateTime

type Query {

feed: [Post!]!

filterPosts(searchString: String): [Post!]!

post(id: ID!): Post

}

type Mutation {

signupUser(email: String!, name: String): User!

createDraft(title: String!, content: String, authorEmail: String!): Post!

deletePost(id: ID!): Post

publish(id: ID!): Post

}

type Post {

id: ID!

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean!

title: String!

content: String

author: User!

}

type User {

id: ID!

email: String!

name: String

posts: [Post!]!

}

```

</details>

### GraphQL server with authentication and permissions

<details><summary>View datamodel and GraphQL schema</summary>

`datamodel.prisma`:

```graphql

type Post {

id: ID! @id

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean! @default(value: "false")

title: String!

content: String

author: User!

}

type User {

id: ID! @id

email: String! @unique

password: String!

name: String

posts: [Post!]!

}

```

`schema.graphql`:

```graphql

scalar DateTime

type Query {

me: User

feed: [Post!]!

filterPosts(searchString: String): [Post!]!

post(id: ID!): Post

}

type Mutation {

createDraft(title: String!, content: String): Post!

deletePost(id: ID!): Post

publish(id: ID!): Post

signup(email: String!, password: String!, name: String): AuthPayload!

login(email: String!, password: String!): AuthPayload!

}

type AuthPayload {

token: String!

user: User!

}

type Post {

id: ID!

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean!

title: String!

content: String

author: User!

}

type User {

id: ID!

email: String!

name: String

posts: [Post!]!

}

```

</details>

### GraphQL server with realtime GraphQL subscriptions

<details><summary>View datamodel and GraphQL schema</summary>

`datamodel.prisma`:

```graphql

type Post {

id: ID! @id

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean! @default(value: "false")

title: String!

content: String

}

```

`schema.graphql`:

```graphql

scalar DateTime

type Query {

feed: [Post!]!

filterPosts(searchString: String): [Post!]!

post(id: ID!): Post

}

type Mutation {

createDraft(title: String!, content: String): Post!

deletePost(id: ID!): Post

publish(id: ID!): Post

}

type Subscription {

posts: Post

}

type Post {

id: ID!

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean!

title: String!

content: String

}

```

</details>

### REST API

<details><summary>View datamodel and API operations</summary>

`datamodel.prisma`:

```graphql

type User {

id: ID! @id

email: String! @unique

name: String

posts: [Post!]!

}

type Post {

id: ID! @id

createdAt: DateTime!

updatedAt: DateTime!

published: Boolean! @default(value: "false")

title: String!

content: String

author: User!

}

```

### API

#### `GET`

- `/post/:id`: Fetch a single post by its `id`

- `/feed`: Fetch all \_published\_ posts

- `/filterPosts?searchString={searchString}`: Filter posts by `title` or `content`

#### `POST`

- `/post`: Create a new post

- Body:

- `title: String` (required): The title of the post

- `content: String` (optional): The content of the post

- `authorEmail: String` (required): The email of the user that creates the post

- `/user`: Create a new user

- Body:

- `email: String` (required): The email address of the user

- `name: String` (optional): The name of the user

#### `PUT`

- `/publish/:id`: Publish a post by its `id`

#### `DELETE`

- `/post/:id`: Delete a post by its `id`

</details>

### CLI TODO app

<details><summary>View datamodel</summary>

`datamodel.prisma`:

```graphql

type Todo {

id: ID! @id

title: String! @unique

createdAt: DateTime!

}

```

</details> -->

## Ways to contribute

### Adding a missing example

The easiest way to contribute is by adding a missing example. Check [this](https://github.com/prisma/prisma-examples/issues/311) GitHub issue to see which examples are currently missing. When adding a new example, please use the suggested [example structure](#example-structure).

### Adding a new example

Before submitting a PR for a new example, please first open an issue that explains the idea of the example and specifies what it will look like (e.g. how the Prisma datamodel will be defined or what kind of API will be built). It'll then be discussed in the issue whether your example is going to be added to the collection. To accelerate the process, you can ping @nikolasburk in the public [Prisma Slack](https://slack.prisma.io).

Once approved, you can add your example to [list of missing examples](https://github.com/prisma/prisma-examples/issues/311) and start implementing it.

### Improving an existing example

If you find a bug in an example, please feel free to open an issue or submit a PR so the bug gets fixed. If you want to make structural changes to an existing example (e.g. changing the datamodel or the API operations), please open an issue about this first where the changes can be discussed. To accelerate the process, you can ping @nikolasburk in the public [Prisma Slack](https://slack.prisma.io).

Once approved, you can go ahead and implement the changes.

### Improving a README

The READMEs for all projects are being auto-generated based on the templates located in [`./github/readmes`](./github/readmes). If you find a typo or other parts of the README that should be improved, please do not edit the README directly but instead add your changes to the corresponding template.

For example, if you found an issue in the `node/graphql` README, do not edit the [`./node/graphql/README.md`](./node/graphql/README.md) file but instead add your changes to [.`/.github/readmes/node/graphql/README.md`](./.github/readmes/node/graphql/README.md). Then build the READMEs using our custom script:

```

cd .github/tests

yarn build-readmes

```