Ruby on Rails GraphQL API Project Structure

GraphQL API with graphql-ruby. Type-safe schema, resolvers, and subscriptions.

#rails #ruby #graphql #api #backend

PNG PDF Copy Prompt

Project Directory

```
myapi/
    Gemfile
    Gemfile.lock
    Rakefile
    config.ru
  app/
    application_controller.rb
    graphql_controller.rb  Single endpoint
  models/
    application_record.rb
    user.rb
    post.rb
  graphql/  GraphQL schema
    myapi_schema.rb  Root schema
    types/  GraphQL types
      base_object.rb
      query_type.rb  Root query
      mutation_type.rb  Root mutation
      user_type.rb
      post_type.rb
      base_input_object.rb
    mutations/
      base_mutation.rb
      create_user.rb
      create_post.rb
      update_post.rb
    resolvers/  Query resolvers
      base_resolver.rb
      users_resolver.rb
      posts_resolver.rb
    loaders/  Batch loading
      record_loader.rb
  services/
    user_service.rb
  config/
    application.rb
    routes.rb
    database.yml
    environments/
    initializers/
      cors.rb
  db/
  spec/
    graphql/  GraphQL query t…
    models/
    factories/
  lib/
  bin/
```

Why This Structure?

graphql-ruby brings GraphQL to Rails with Ruby-native type definitions. Define types in Ruby classes, wire up resolvers, and get a type-safe API. Includes batching (graphql-batch) to solve N+1 queries.

Key Directories

app/graphql/types/ - GraphQL object types matching your models

app/graphql/mutations/ - Mutation classes for create/update/delete

app/graphql/resolvers/ - Query resolvers with pagination, filtering

app/graphql/loaders/ - Batch loaders to prevent N+1

GraphQL Type with Dataloader

```ruby
# app/graphql/types/user_type.rb
class Types::UserType < Types::BaseObject
  field :id, ID, null: false
  field :email, String, null: false
  field :posts, [Types::PostType], null: false

  def posts
    dataloader.with(Sources::ActiveRecord, Post)
      .load_all(object.post_ids)
  end
end
```

Getting Started

1. `rails new myapi --api`
2. `bundle add graphql`
3. `rails generate graphql:install`
4. `rails generate graphql:object User`
5. Visit /graphiql for playground

When To Use This

- Frontend needs flexible data fetching
- Multiple clients with different data needs
- Mobile apps wanting minimal payloads
- Complex nested data relationships
- When REST endpoints are proliferating

Trade-offs

N+1 queries - Requires dataloader/batch loading setup

Caching complexity - HTTP caching harder than REST

Learning curve - GraphQL concepts for whole team

Testing Strategy

spec/graphql/ - Test queries and mutations directly

Schema snapshots - Detect breaking schema changes

Resolver tests - Unit test resolvers with mocked context