Hanami Framework Cheatsheet

A quick reference guide to the Hanami framework, covering essential commands, configurations, and concepts for building robust web applications.

Project Setup & Core Concepts

Project Initialization

Creating a new Hanami project:

```
hanami new my_project
cd my_project
```

This command generates a basic Hanami project structure.

Starting the development server:

```
bundle exec hanami dev
```

Launches the Hanami development server, usually on `localhost:2300`.

Project Structure Overview:

- `apps/`: Contains individual Hanami applications (e.g., `web`).
- `config/`: Configuration files for the entire project.
- `db/`: Database-related files (migrations, schema).
- `lib/`: Core application logic and entities.

Application Architecture

Slice	A modular component encapsulating specific functionality within an application.
Actions	Handle incoming HTTP requests and orchestrate the response. Similar to controllers in other frameworks.
Views	Responsible for rendering the response. They prepare data for templates.
Repositories	Interact with the database. Provide an abstraction layer for data access.
Entities	Represent domain objects. They encapsulate data and business logic.

Routing & Controllers (Actions)

Defining Routes

Basic Route Definition:

Inside `config/routes.rb`:

```
slice :main, at: "/" do
  get "/", to: "home#index"
end
```

This maps a GET request to `/` to the `index` action of the `HomeController`.

Route Shorthands:

```
get "/articles", to: "articles.index"
post "/articles", to: "articles.create"
put "/articles/:id", to: "articles.update"

delete "/articles/:id", to: "articles.destroy"
```

Resources:

```
resources :articles
```

Automatically generates routes for common CRUD operations.

Creating Actions

Action Class Structure:

```
# in apps/web/controllers/home/index.rb
module Web::Controllers::Home
  class Index
    include Web::Action

    def call(params)
      # params contains request parameters
      # Perform logic here

      # Set response data
      @message = "Hello, Hanami!"
    end
  end
end
```

Exposing Data to Views:

Instance variables set in the `call` method are automatically available in the corresponding view.

Handling Parameters:

Request parameters are accessible through the `params` object.

Views & Templates

View Components

Basic View Structure:

```ruby
# in apps/web/views/home/index.rb
module Web::Views::Home
  class Index
    include Web::View

    def message
      raw(context[:message])
    end
  end
end
```

Templates:

Located in `apps/web/templates/home/index.html.erb` (or other template engines).

Access data exposed by the view using instance variables or the `context`.

Partials:

Create reusable template snippets.

```ruby
<%= partial 'shared/footer' %>
```

Template Engines

ERB	Embedded Ruby, the default template engine.
Slim	A fast and lightweight template engine with a clean syntax.
Haml	Another popular template engine with a concise syntax.

Models & Repositories

Defining Entities

Entity Structure:

```ruby
# in lib/my_project/entities/article.rb
class Article < Hanami::Entity
end
```

Entities represent domain objects. They encapsulate data and business logic.

Attributes:

Attributes are defined through database schema.

Example: `:id`, `:title`, `:content`

Repositories

Repository Structure:

```ruby
# in lib/my_project/repositories/article_repository.rb
class ArticleRepository < Hanami::Repository
end
```

Common Operations:

- `ArticleRepository.new.find(id)`: Finds an entity by ID.
- `ArticleRepository.new.create(data)`: Creates a new entity.
- `ArticleRepository.new.update(id, data)`: Updates an existing entity.
- `ArticleRepository.new.delete(id)`: Deletes an entity.

Querying:

```ruby
ArticleRepository.new.where(title: 'Example').all
```