**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты разрабатываемой системы и выработав список требований необходимых для разработки системы, разбиваем систему на функциональные блоки(модули). Это необходимо для обеспечения гибкой архитектуры. Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В разрабатываемом веб-приложении можно выделить следующие блоки:

* блок интерфейса пользователя;
* блок модели базы данных;
* база данных веб-приложения;
* блок регистрации;
* блок авторизации;
* блок ядра веб-приложения.

В управляющей программе можно выделить следующие блоки:

* блок ядра управляющей программы;
* блок определения движения;
* блок работы с данными на SD-карте;
* блок получения изображения с камеры;
* блок взаимодействия с веб-приложением.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.047 C1.

Каждый модуль выполняет свою задачу. Чтобы система работала каждый модуль взаимодействует с другими модулями путем обмена данными, используя различные форматы и протоколы.

Рассмотрим функциональные блоки веб-приложения.

*Блок интерфейса пользователя* является клиентской частью веб-приложения. Данный блок представляет собой совокупность средств, при помощи которых пользователь взаимодействует с приложением через браузер. Для построения интерфейса используется технология Haml. Haml – язык разметки для упрощенной генерации HTML, который компилируется в HTML. Он позволяет представлять данные, а также обрабатывать ввод пользователя. Официальная реализация этого языка написана на Ruby.

*Блок модели базы данных* представляет из себя набор моделей определенных библиотекой ActiveRecord в фреймворке RubyOnRails, по сути является ключевым модулем на стороне сервера, содержит в себе схему базы данных и отвечает за генерацию запросов к СУБД с последующим «маппингом» полученных данных в привычные и удобные объекты языка Ruby.

*База данных веб-приложения* включает данные, используемые веб-приложением. При реализации использовалась база данных PostgresSQL. Это свободно распространяемая объектно-реляционная система управления базами данных наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

Основными достоинствами PostgresSQLявляются:

* надежность (полное соответствие принципам ACID - атомарность, непротиворечивость, изолированность, сохранность данных);
* производительность (основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе);
* расширяемость (означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов);
* поддержка SQL;
* богатый набор типов данных;
* простота использования;
* безопасность данных.

*Блок регистрации* отвечает за регистрацию пользователей и сохранение их данных в базу данных веб-приложения.

При реализации использовался плагин Devise. Этот плагин предоставляет решение с весьма гибкими настройками для задач регистрации и аутентификации пользователей.

*Блок авторизации* отвечает за авторизацию пользователей и разделение прав доступа к ресурсам веб-приложения.

При реализации использовался плагин Devise, рассмотренный выше, а также плагин Cancancan. Сancancan позволяет устанавливать различные уровни доступа для пользователей, таким образом ограждая определнные ресурсы от несанкционированного доступа.

*Блок ядра веб-приложения* представляет встроенное RubyOnRails приложение.

Фреймворк RubyOnRails написан с помощью языка программирования Ruby[7].

Ruby – [интерпретируемый](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80) [язык программирования высокого уровня](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Обладает независимой от [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) реализацией [многопоточности](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C), строгой [динамической типизацией](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [«сборщиком мусора»](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D1%89%D0%B8%D0%BA_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0) и многими [другими возможностями](https://ru.wikipedia.org/wiki/Ruby#.D0.92.D0.BE.D0.B7.D0.BC.D0.BE.D0.B6.D0.BD.D0.BE.D1.81.D1.82.D0.B8_Ruby), поддерживающими много разных [парадигм программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), прежде всего [объектно-ориентированную](https://ru.wikibooks.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

RubyonRails предоставляет из себя архитектуру [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller) для [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), а также обеспечивает их интеграцию с [веб-сервером](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) и сервером базы данных. RubyonRails определяет следующие принципы разработки приложений, помогающие разработчику в создании элегантных программных решений, усвоенные сообществом разработчиков:

* предоставляет механизмы [повторного использования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0), позволяющие минимизировать дублирование кода в приложениях (принцип [Don’t repeat yourself](https://ru.wikipedia.org/wiki/Don%E2%80%99t_repeat_yourself));
* [по умолчанию](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE_%D1%83%D0%BC%D0%BE%D0%BB%D1%87%D0%B0%D0%BD%D0%B8%D1%8E) используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration);
* основными компонентами приложений RubyonRails являются модель, представление и контроллер;
* RubyonRails использует [REST](https://ru.wikipedia.org/wiki/REST)-стиль построения веб-приложений.

Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных (таких как каталог продуктов или список заказов). Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику.

Для хранения объектов модели в реляционной СУБД по умолчанию в Rails использована библиотека [ActiveRecord](https://ru.wikipedia.org/wiki/ActiveRecord). Конкурирующий аналог – DataMapper. Существуют плагины для работы с [нереляционными базами данных](https://ru.wikipedia.org/wiki/NoSQL), например Mongoid для работы с [MongoDB](https://ru.wikipedia.org/wiki/MongoDB).

Представление создает [пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) с использованием полученных от контроллера данных. Представление также передает запросы пользователя на манипуляцию данными в контроллер (как правило, представление не изменяет непосредственно модель).

В RubyonRails представление описывается при помощи [шаблонов](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD) ERB. Они представляют собой файлы [HTML](https://ru.wikipedia.org/wiki/HTML) с дополнительными включениями фрагментов кода [Ruby](https://ru.wikipedia.org/wiki/Ruby) (EmbeddedRuby или ERB). Вывод, сгенерированный встроенным кодом Ruby, включается в текст шаблона, после чего получившаяся страница HTML возвращается пользователю. Кроме ERB возможно использовать ещё около 20 шаблонизаторов, в том числе [Haml](https://ru.wikipedia.org/wiki/Haml) и Slim.

Контроллер в Rails – это набор логики, запускаемой после получения HTTP-запроса сервером. Контроллер отвечает за вызов методов модели и запускает формирование представления.

Вокруг Rails сложилась большая экосистема [плагинов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD) – подключаемых «[гемов](https://ru.wikipedia.org/wiki/RubyGems)» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) gem), некоторые из них со временем были включены в базовую поставку Rails, например [Sass](https://ru.wikipedia.org/wiki/Sass) и [CoffeeScript](https://ru.wikipedia.org/wiki/CoffeeScript), другие же, хотя и не были включены в базовую поставку, являются фактическим стандартом для большинства разработчиков, например, средство [модульного тестирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) RSpec.

Теперь рассмотрим функциональные блоки управляющего приложения.

*Блок ядра управляющей программы* является основным блоком управляющей программы. Данный блок представляет собой приложние, написанное на языке С++.

С++ − это компилируемый язык программирования со статической типизацией. Язык поддерживает несколько парадигм программирования и имеет богатую стандартную библиотеку. С++ сочетает в себе свойства как высокоуровневых, так и низкоуровневых языков программирования.

Блок ядра управляющей программы отвечает за инициализацию переменных, установку режимов входов платы, а также управляет остальными функциональными блоками управляющей программы.

*Блок определения движения* отвечает за обработку сигнала, поступающего с датчика движения, также уведомляя об этом блок получения изображения с камеры.

*Блок получения изображения с камеры* позволяет послать сигнал камере, при котором она сделает снимок. Этот снимок обрабатывается данным блоком и пересылается в блок работы с данными на SD-карте.

*Блок работы с данными на SD-карте* реализует операции записи данных на SD-карту и чтения с нее.

*Блок взаимодействия с веб-приложением* позволяет отправлять фотографии, полученные с камеры и хранящиеся на SD-карте на сервер нашего веб-приложения.