Документация

# Общие положения

В этом разделе описываются общие положения

**Определение, Утверждение:**

Таким заголовком обозначаются различного рода определения и утверждения, используемые в системе.

*Примечание, Совет:*

Таким заголовком обозначаются дополнительные сведения, раскрывающие тонкости содержания раздела и/или рекомендации

*Важно:*

Таким заголовком обозначаются важные предупреждения, в том числе и критические. Не соблюдение которых, может привести к потере данных или работоспособности.

Инструкции, конструкции

Таким текстом пишутся конкретные конструкции языка, системы и примеры программного кода.

## Принятая нумерация версий системы

X.X.X или R.A.B

R (Release) - Первая цифра обозначает номер релиза. Итерация этого порядка привязывается к завершению очередного этапа разработки. Как правило, это происходит при завершении разработки каких-либо новых функциональных возможностей (модулей) или фундаментальных изменений самой системы. Или когда их количество и качество становится слишком большим.

A (Advanced) – Вторая цифра обозначает номер дополнения к релизу. Итерация этого порядка происходит при разработке нового или доработке и расширении ранее реализованного функционала.

Эта цифра обнуляется во время итерации первой цифры R.

B (Bugfix) – Третья цифра обозначает номер исправленной ошибки в работающем проекте. Итерация этого порядка происходит при исправлении выявленных ошибок в уже работающем проекте.

Эта цифра обнуляется во время итерации первой цифры R

## ТД (технические директивы) и стиль кодирования

1)

Имена всех переменных, типов, функций, методов формируются из заглавный и строчных букв без использования знаков «\_», «-» и прочих разделителей.

Первая буква всех (рабочих внутри пакетных) переменных, как правило, строчная.

(под переменными в данном случае понимается любая переменная, включая типы и исключая константы)

Имена констант состоят из заглавных букв, и используют в имени символ нижнего подчёркивания в качестве разделителя словоформ. (Приватные константы мы не используем)

Все имена формируются из трёх ключевых понятий:

Функциональное назначение переменной (получение, сохранение, вычисление…).

Имя нарицательное (с чем собственно работаем – item, goods, news, article…).

Тип хранимых данных в переменной (список, хеш, число...).

Наличие все трёх составляющих и их порядок опциональны.

*Рекомендуется также при формировании имён учитывать их прикладное назначение.*

*К примеру, загрузка и сохранение объектов в БД в парадигме ОРМ и приложения используют в именах ключевые слова Load и Save. А загрузка и сохранение простого массива данных в БД в парадигме БД используют в именах ключевые слова Select и Update соответственно. (В данном случае рассматривалась секция (словоформа) функционального назначения)*

БД

Имена таблиц и их полей подчиняются правилу именования переменных.

Со следующими отличиями:

Все имена начинаются с заглавной буквы

Символ “\_” используется только в полях реализующих связи

Name, IsAccess, AccessControllers\_Id

2)

Функции конструкторы начинаются с префикса ‘New’. Или целиком состоят из него.

Функции, запускаемые как отдельные программы или запускающие внутри себя таковые начинаются, заканчиваются или целиком состоят из словоформы ‘G(g)o’.

Интерфейсы заканчиваются или целиком состоят из суффикса ‘Face’.

3)

// Комментарий - заголовочный комментарий (отделяется дополнительно пустой строкой)

Для правильной работы IDE и документирования заголовочный комментарий метода или функции должен начинаться с имени метода или функции

// дополнительный комментарий (не отделяется дополнительно пустой строкой)

doc.go либо основной файл пакета (смотри 1.4)

/\*

Описание пакета. Заголовок.

Описание пакета.

@author Автор <Мыло автора >

@version $Id$

@link http://www.domain.com/

@copyright <COPYRIGHT>

@license http://www.domain.com/license/

\*/

Многострочный комментарий разрешен только в заголовках файлов пакета перед всем остальным кодом.

4) Импортирование пакетов

Импортируемые пакеты группируются в 3 группы, разделенные пустой строкой.

Первая группа – стандартные пакеты Го

Вторая группа – сторонние пакеты (3 лиц)

Третья группа - родные пакеты (пакеты проекта)

Пример:

Import (

“time”

“mymysql”

“lib/mysql”

)

## Словарь и термины

**Модель** – Значение структуры определенного типа, описывающей конкретный источник в БД (имя структуры – имя источника, поле в источнике – поле в структуре). С набором методов по работе с данными источника.

**Голая модель** – Это модель, не имеющая ни одного персонального метода.

**Тип** – Это структура, описывающая конкретный источник в БД. Также содержит набор сценариев.

**Сценарий** – Список конфигураций для полей. Обеспечивает настраиваемую работу с данными источника.

**Источник, БД** – Под источником понимается абстрактное хранилище конкретных данных.

Это может быть таблица в реляционной БД Mysql, или коллекция в Mongo DB или просто файл.

**Поле** – Свойство структуры любого типа в любом типе. Поле в таблице БД.

**Пакет** – Это пакет в Го. Обобщенное название для: компонент, контроллер, модель, библиотека, тип

**Демон** – Программа, работающая параллельно от основной программы.

**Система** – Совокупность программного кода реализующего системную работу программы в принципе и его базовый и абстрактный функционал. (Обработка запросов, работа контроллеров, отдача результата)

**Приложение** – Совокупность программного кода реализующего прикладную работу программы в целом и частности.

**Список** – срез ([]string).

**Хеш** – именованный массив (map[string]string).

**Ядро** – Пакет ‘core’ содержит базовый необходимый набор данных и функционал для обеспечения работоспособности приложения в целом.

**Компонент** - Это логически законченный системный функционал решающий определенный круг связанных задач. Неотъемлемая часть приложения. (Ядро, базовая модель, базовый тип, базовый контроллер)

**Модуль** – Совокупность программного кода решающий конкретную задачу проекта. Эта совокупность как правило представлена опциональным набором типов, моделей и контроллеров размещающихся в папках `app` и `types`.

## Файловая структура пакетов

Пакет формируется из файлов разбитых по функциональному признаку. Пример:

…/**packagename**/

**packagename.go**\* Основной файл пакета. Содержит основной или единственный функционал пакета. Может включать в себя соответственно все нижеописанные части.

**functionalname.go** Файл содержащий логически законченный программный код, реализующий конкретную прикладную задачу пакета.

**doc.go** Документация и примеры использования пакета.

Блоки документации также могут располагаться в других файлах пакета.

**config[…].go** Конфигурации пакета.

**init[…].go** Инициализации пакета.

(Если есть инициализации, то конфигурационные данные предпочтительно размещаются в них)

\* - предпочтительно обязательные файлы составляющие пакет.

Основной файл пакета необходимо назвать по имени пакета, в основном файле пакета находятся основные вызываемые извне функции или методы.

Платформ зависимый программный код размещается в одноименных функциональных файлах с добавлением суффикса платформы (\_windows, \_darwin, \_linux).

В каждом платформа зависимом файле обязательно должен находиться один и тот же набор функций или методов.

Также в пакете могут присутствовать файлы тестирования (\_test)

## Файловая структура системы

app/ Контроллеры и модели приложения (разбитые по модулям)

access/ Модуль содержащий контроллеры и модели

users/ Контроллер модуля

lib/ Родные библиотеки

core/ Ядро приложения (система)

route/ Роутер. Веб-сервер.

factor/ Приложение. Управляющий контроллер. (Принимает и обрабатывает запросы)

daemon/ Контроллеры работающие в фоновом режиме (как отдельные программы).

types/ Типы. Проекция БД реализованная в структурах.

сторонние библиотеки (кладутся в корень src проекта)

Здесь нарисовать картинку зависимостей пакетов кто кого импортирует либо может импортировать

## Для разработчика

### Ведение логов и журнала

Логирование и журналирование работы приложения производить только в точке входа (`application.go`), ядре (`core`), контроллерах и демонах. В остальных местах лишь формировать ошибки и сообщения (errors и подобные)

### Инкапсуляция пакетов

По возможности максимально “прятать” внутри всю служебную работу и необходимые для этого данные пакета. “Маша с Дубровским общаются через дупло”

### Конфиги

В конфигурации приложения размещаются только системная конфигурация влияющая на всю работу системы в целом. Частная конфигурация отдельных функциональностей размещается в соответствующих модулях (го) либо БД.

### Проекция источников (таблиц БД ) на структуры в программе

Для источников содержащих записи сущностей создаются модели, типы и контроллеры.

(Могут содержаться в памяти в чистом виде)

Для источников содержащих отношения сущностей (кросс связь со свойствами) создаются модели и типы.

(Могут содержаться в памяти в чистом виде) ~~(В памяти в чистом виде не хранятся)~~

Для источников содержащих связь многие ко многим (кросс таблицы) не создается ничего.

# Логика работы системы

Здесь описывается весь логический порядок работы системы от начала до конца (с высоты птичьего полета).

Также некоторые важные особенности реализации (2.3 и далее до 3).

## Запуск приложения

### Чтение и инициализация параметров командной строки

### Поиск конфигурационного файла

### Чтение, инициализация и сохранение конфигурации в `core`

### Инициализация (создание если их нет) папок приложения

### Инициализация используемых в приложении библиотек

### Проверка и настройка железа (память, процессоры)

### Проверка соединений с БД

### Загрузка системных данных из БД

### Создание PID файла и блокировка на запись

### Запуск роутеров (веб серверов) Работа приложения в штатном режиме

## Работа приложения в штатном режиме CMF

…

## Работа приложения в штатном режиме FrameWork

…

## Реализация абстрактных горизонтальных прав

Поля связи могут выступать в роли УП если они сп­­роектированы в таблице пользователей (`core\_users`).

`Cars\_Marka\_Id` (пользователь работает только с моделями определенной марки)

Кроме того идентификатор пользователя становится условием самого пользователя (`Core\_Users\_Id`)

Структура хранения пользователей каталогизирована. Это позволяет реализовывать реальную и необходимую структуру компании в частности и целом. Впрочем, это относится ко всем пользователям проекта. Условие пользователя (если оно установлено) включает самого пользователя и его подчиненных (связанных с ним родительской связью) на один уровень вниз.

Такое решение позволяет организовывать систему прав как между различными подразделениями (группами) по функциональному назначению, так и в внутри одно подразделения дифференцировать согласно ее иерархии.

Тонкости реализации:

`Core\_Users\_Id`

`Cars\_Marka\_Id`

УП определяется по наличию полей связи в структуре таблицы и статуса УП (`core\_users`.`IsCondition`) а также установленным в них значениям.

При обнаружении одноименных полей связи в целевой таблице УП начинают работать.

# Компоненты и их взаимодействия между собой:

Здесь описывается компоненты системы, из зависимости и взаимодействия между собой

## core (Ядро)

### Назначение

### Реализация

### Зависимости и взаимодействия

## core/route (Роутер)

### Назначение

### Реализация

### Зависимости и взаимодействия

## core/factor (Приложение и базовый контроллер)

### Назначение

### Реализация

### Зависимости и взаимодействия

## core/factor (Приложение и базовый контроллер)

### Назначение

### Реализация

### Зависимости и взаимодействия

# Библиотеки

## lib/mysql (работа с БД mysql)

### Назначение

### Реализация

### Зависимости и взаимодействия

## lib/logs (Логирование)

### Назначение

### Реализация

### Зависимости и взаимодействия