Лекция 1
Введение в Web и Шаблонизация

Разработка интернет приложений

Оценивание и сроки

- Экзамен
- 2 рубежных контроля
- Практические задания – закрепление и использование знаний разных дисциплин

- Оценивание – баллы за задания
- Сроки!!!
- Участие в Хакатоне Кафедры ИСиТ (~конец апреля)

Одна тема на весь курс

- Набор требований по каждому заданию + порядок показа
- 8 лабораторных + GitHub + UML + защита + конспект
- ТЗ (Модуль 1)
- ДЗ: три дополнительных задания и готовый Отчет-РПЗ

- Знание браузера, умение использовать необходимые инструменты
- Ответы на вопросы по базовым понятиям и технологиям

Стек технологий

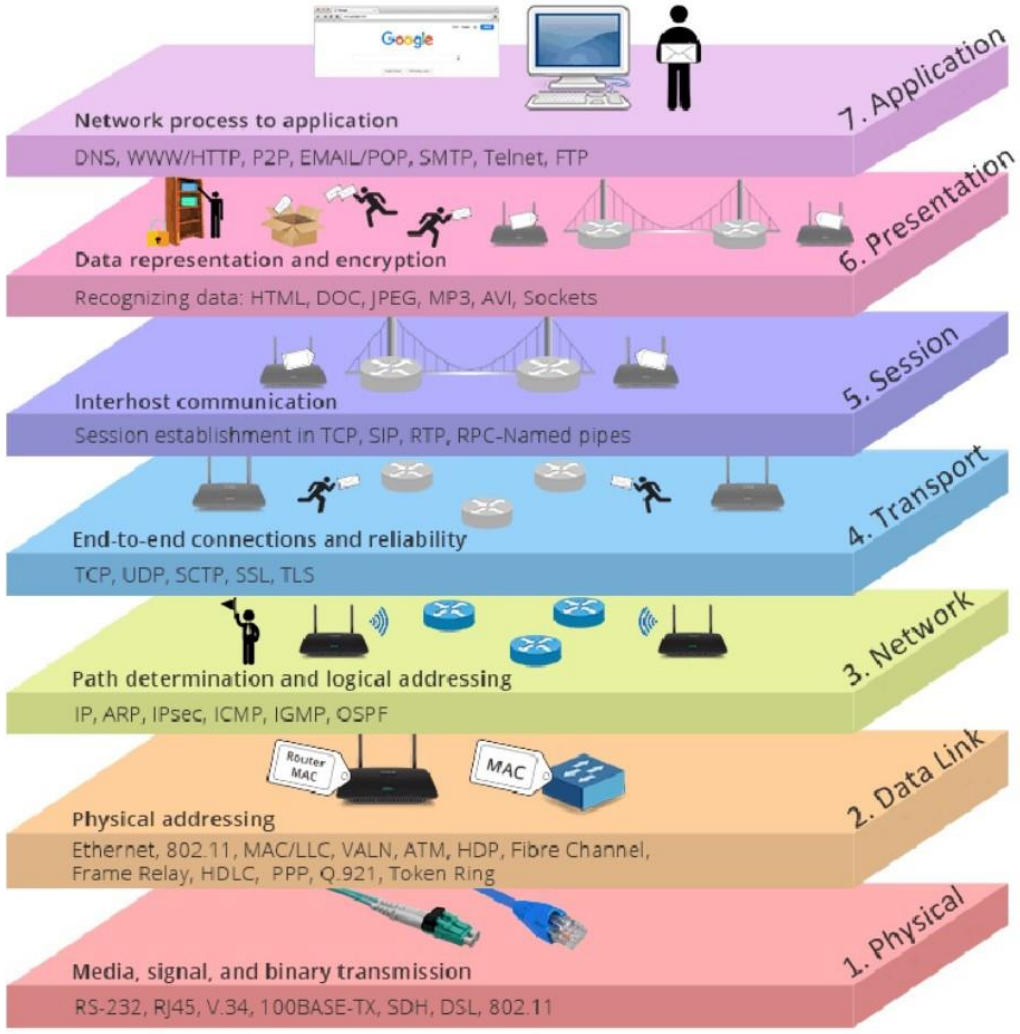
- React (самый популярный в РФ и мире) + Redux + React Bootstrap
- Django или Go. Другой бэкенд только по согласованию с преподавателем
- PostgreSQL

- GitHub - репозитории для фронтенда, бэкенда, нативного приложения. Вы работаете на свое портфолио
- VS Code – основная среда разработки
- Docker

Компьютерные сети. Модель OSI

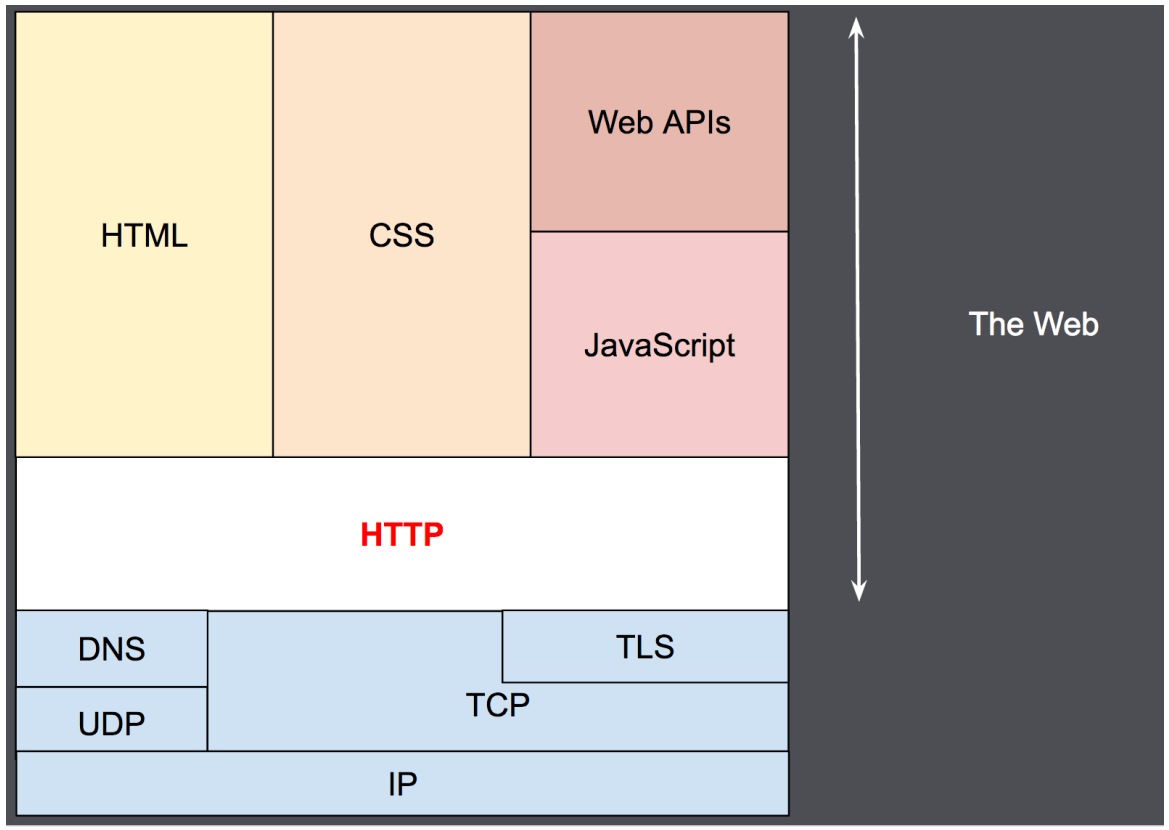
- 7-ми уровневая модель OSI
- Это детальное описание интернета

- Приложения работают на самом высоком 7-ом уровне
- Порты для программ на 4-ом
- IP адреса компьютеров 3 уровень
- Физическая среда передачи на первом уровне



Граница Web/интернет

- Стандарты Web публикуются на сайте веб-консорциума
- https://www.w3.org



Компоненты Web – знать обязательно!

- Тим Бернерс-Ли создал три основных компонента WWW:
- язык гипертекстовой разметки документов HTML (HyperText Markup Language);
- универсальный способ адресации ресурсов URI (Universal [Uniform] Resource Identifier);
- протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol – протокол передачи гипертекста).
- Позже к этим трем компонентам добавился четвертый CGI: исполняемая часть, с помощью которой можно создавать динамические HTML-документы.

HTML

- HTML-HyperText Markup Language.
- В HTML версии 1.0 были реализованы все элементы разметки, связанные с выделением параграфов, шрифтов, стилей и т.п., т.к. уже первая реализация подразумевала графический интерфейс. Важным компонентом языка стало описание гипертекстовых ссылок, графики и обеспечение возможности поиска по ключевым словам.
- В качестве базы для разработки языка гипертекстовой разметки HTML был выбран SGML (Standard Generalised Markup Language – стандартный общий язык разметки). Тим Бернерс-Ли описал HTML в терминах SGML как описывают языки программирования в терминах формы Бекуса-Наура.

URI – схема HTTP

- http:// хост : порт / путь и имя файла ? параметры # якорь гиперссылки

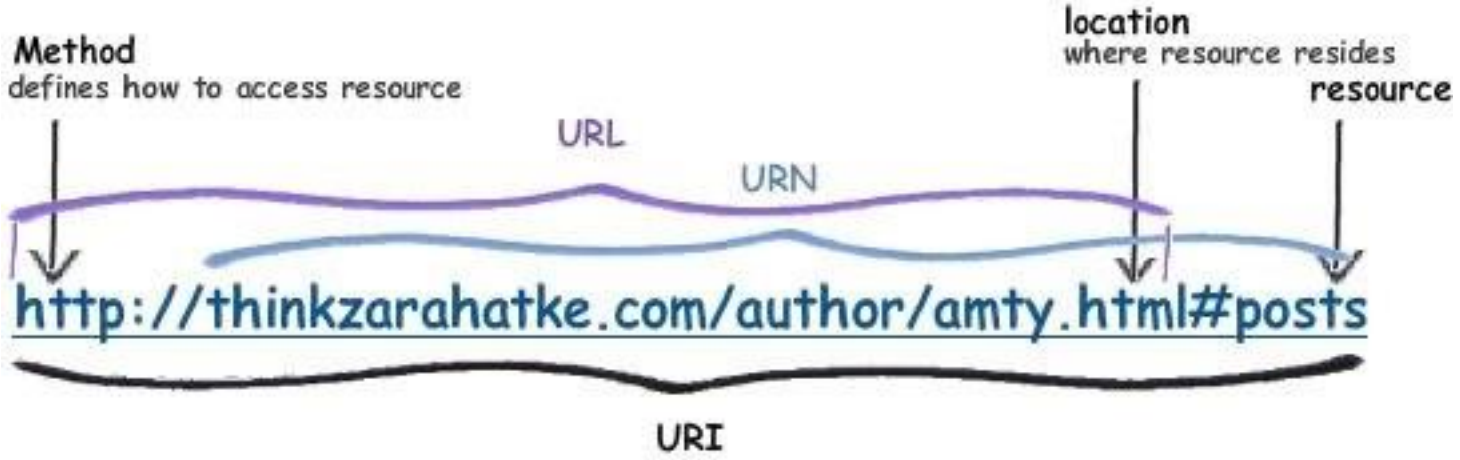
Пример:

http:// 127.0.0.1 :8080/index.html

http://localhost:8080/file.html

http://iu5.bmstu.ru:8080/cat1/cat2/script.asp?param1=1&param2=2#anchor1

- Порт по умолчанию – 80.



HTTP request/response

- Методы

GET, POST, PUT, …

- Коды состояний

200 OK

404 Not Found

- Заголовки

параметр: значение

File Edit View Search Terminal Help

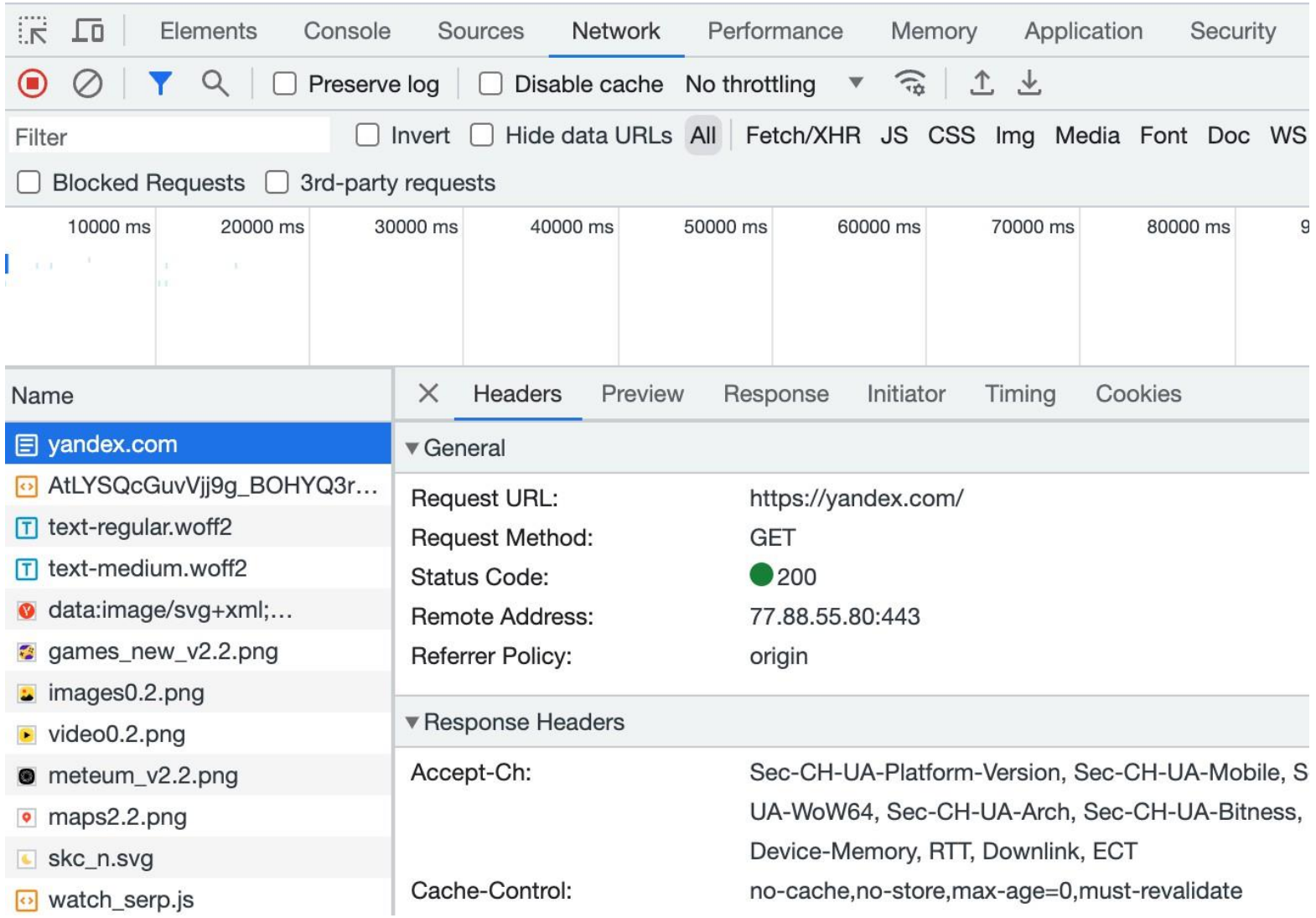
```
[osboxes@osboxes ~]$ telnet iu5.bmstu.ru 80
Trying 195.19.50.252...
Connected to iu5.bmstu.ru.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 09 Nov 2020 08:53:01 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 985
Connection: close
Last-Modified: Fri, 12 Apr 2019 09:22:18 GMT
ETag: "3d9-58651d6d73b52"
Accept-Ranges: bytes

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"><head>
    <title>hoster1.uimp.bmstu.ru &mdash; Coming Soon</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <meta name="description" content="This is a default index page for a new domain."/>
    <style type="text/css">
        body {font-size:10px; color:#777777; font-family:arial; text-align:center;}
        h1 {font-size:64px; color:#555555; margin: 70px 0 50px 0;}
        p {width:320px; text-align:center; margin-left:auto;margin-right:auto; margin-top: 30px }
        div {width:320px; text-align:center; margin-left:auto;margin-right:auto;}
        a:link {color: #34536A;}
        a:visited {color: #34536A;}
        a:active {color: #34536A;}
        a:hover {color: #34536A;}
    </style>
</head>
```

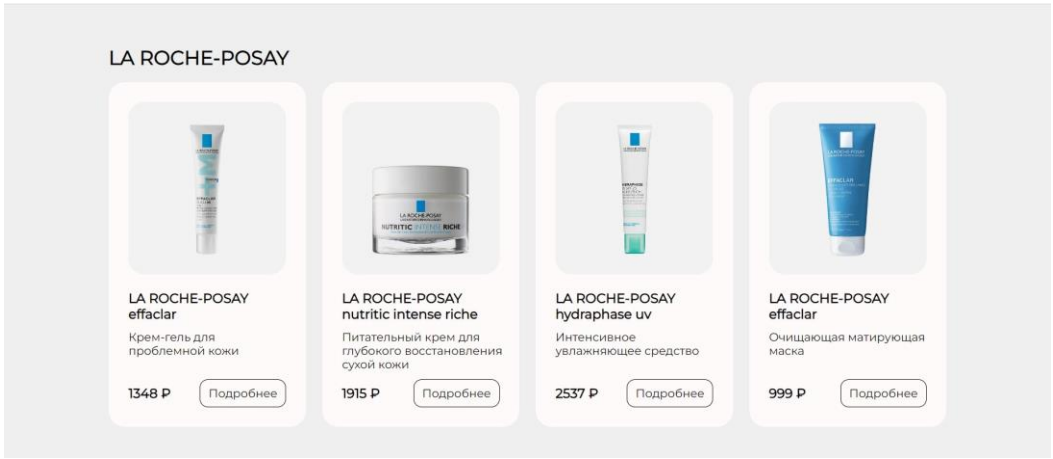
HTTP запросы в браузере

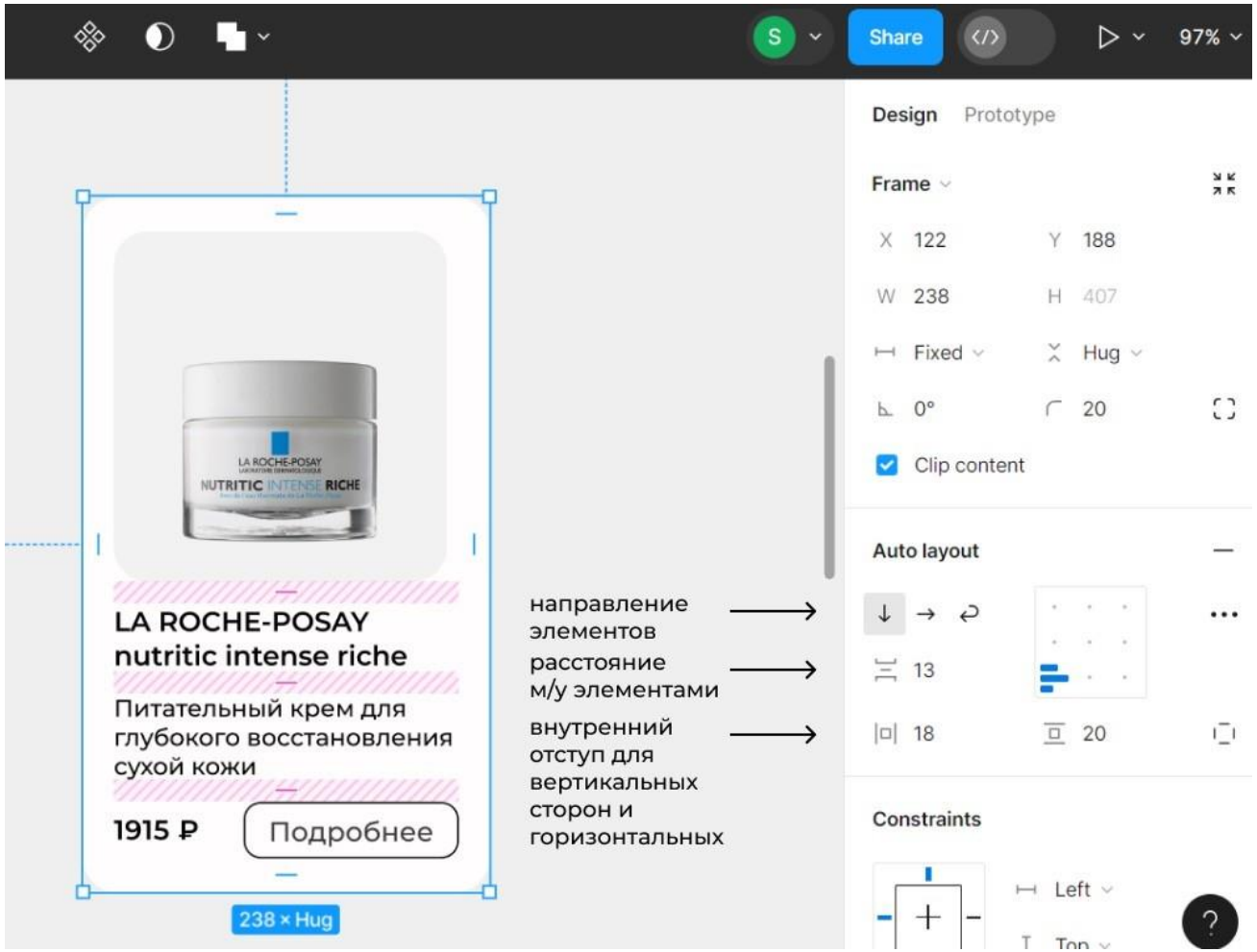
- На вкладке Network отображаются все запросы
- Получение HTML, js, css, изображений, а также AJAX (Fetch/XHR) запросы
- Отображаются заголовки, ответ и тд



Figma/Pixso и CSS

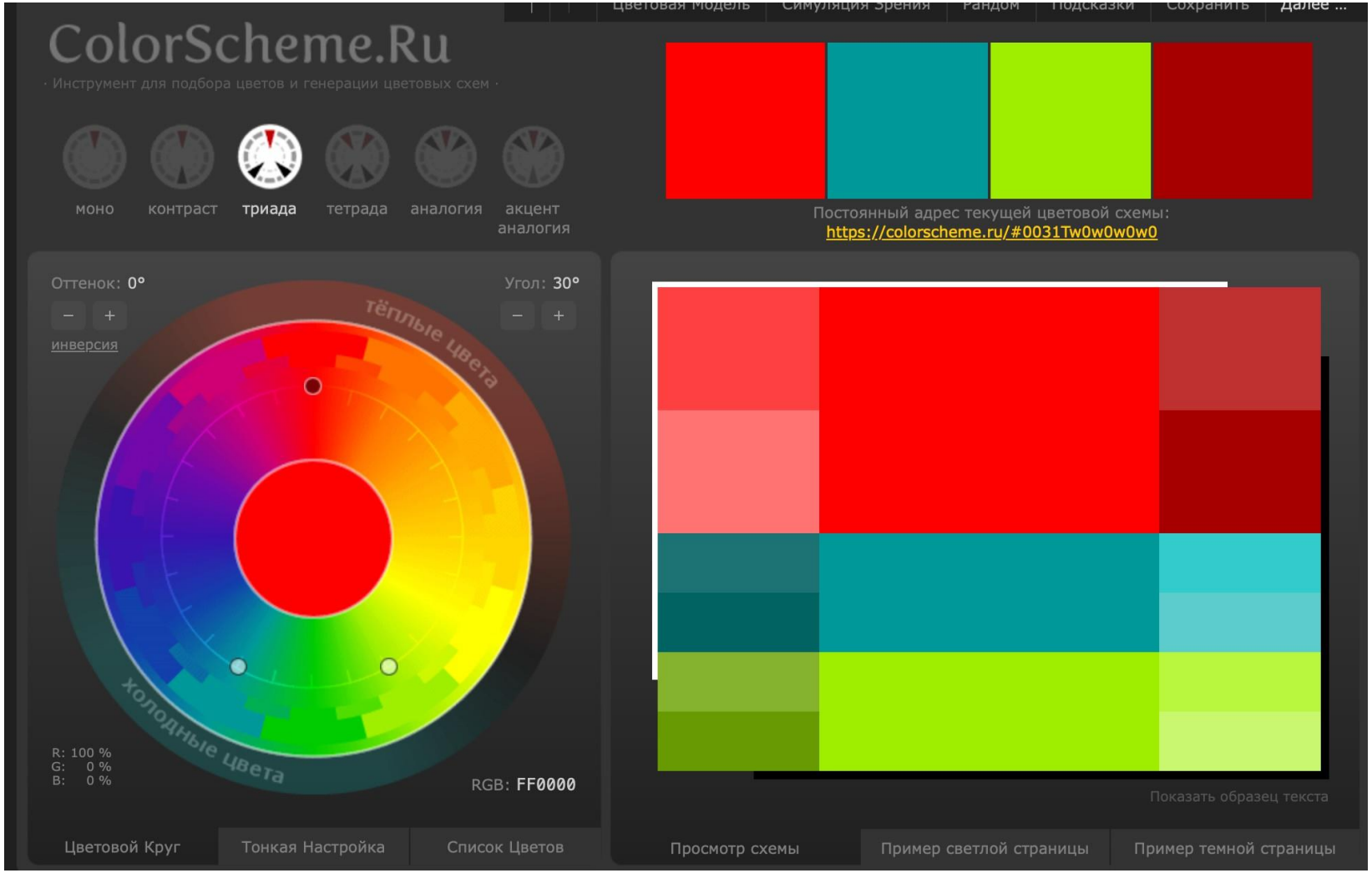
- Дизайн первых 3 страниц приложения вы создаете в Figma/Pixso
- Затем стили ваших карточек вы переносите в CSS вашего проекта





Дизайн приложения

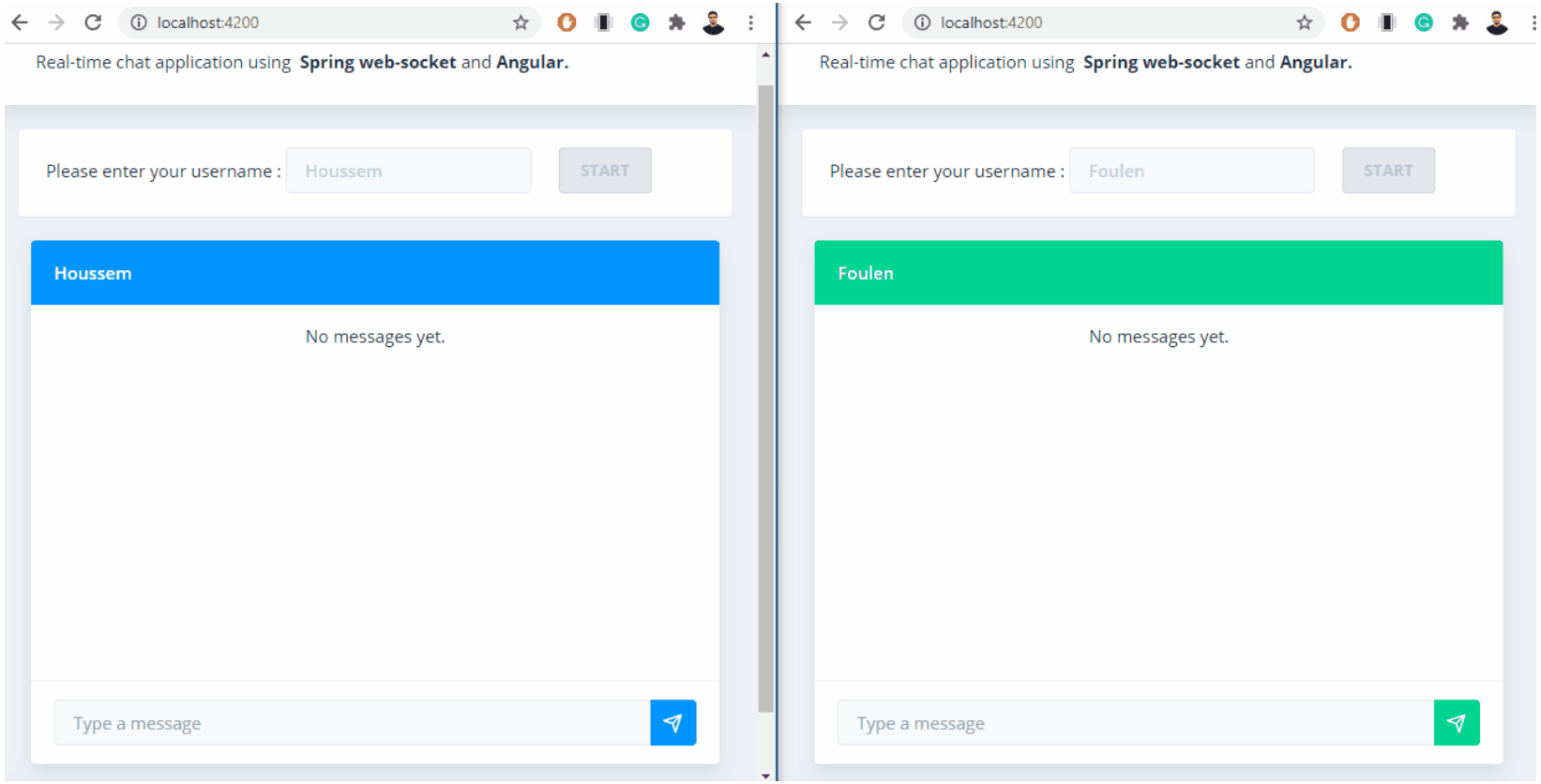
- Работа над дизайном приложения с первого занятия
- Цветовая схема. https://colorscheme.ru



Real-time web

- Ajax
- Push
- WebSocket

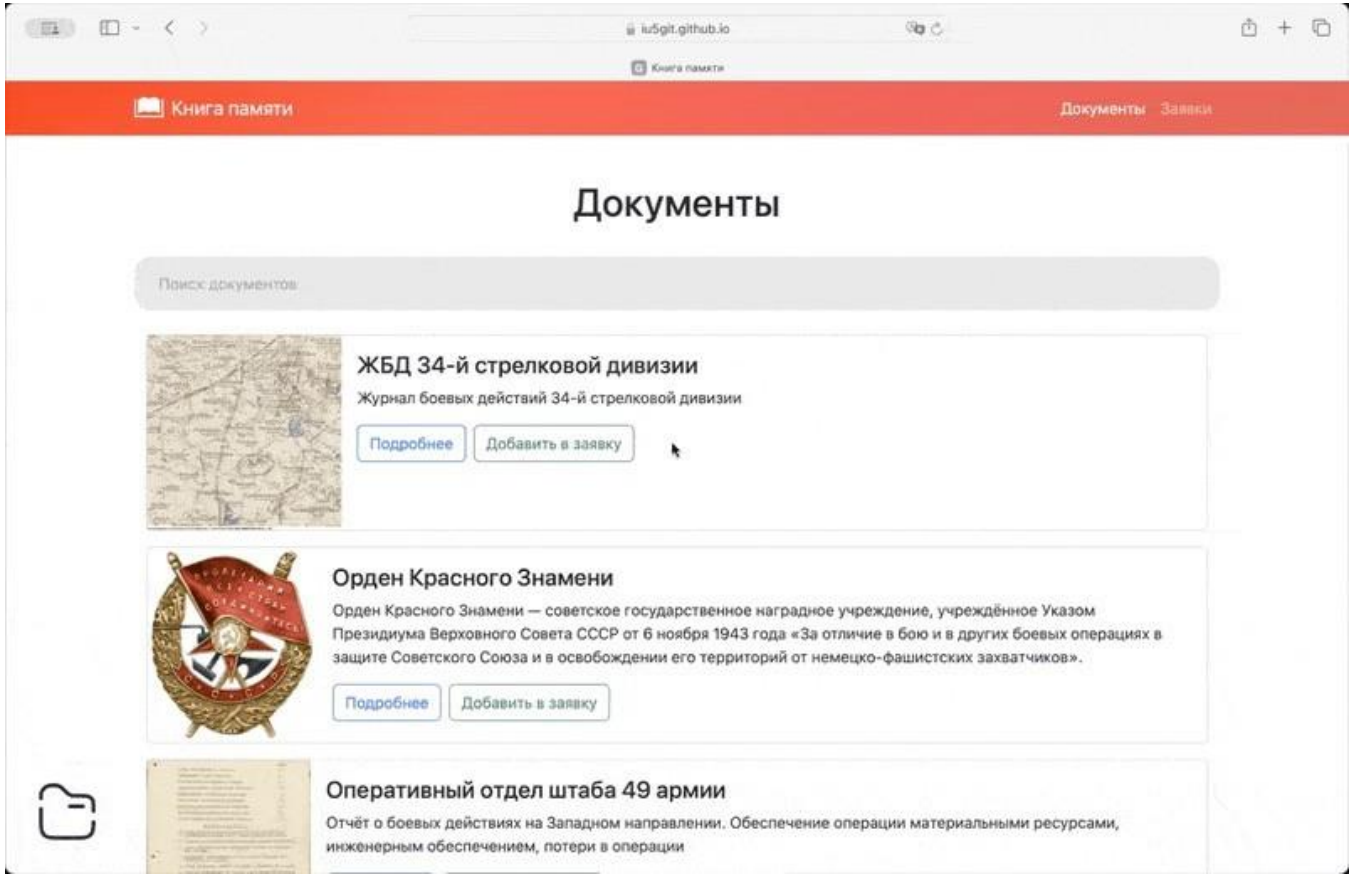
- Подробнее остановимся на курсовой весной



Итоговое приложение курса

- Вам требуется разработать приложение для работы с заявками на услуги по вашей теме
- У всех один и тот же движок
- В этом примере услуги – это документы по ВОВ

- В первой лабораторной нужно реализовать три страницы: все услуги, одна услуга и одна заявка (корзина)
- Пока только просмотр, редактирование добавится позже



Web-фреймворки

- Клиентские фреймворки (Angular, React, Vue)

Предназначены для разработки SPA. Реализуют концепцию «толстого» клиента и «тонкого» сервера. Основная функциональность реализована с использованием JavaScript/TS.

- Серверные фреймворки

Предназначены для разработки приложений на стороне веб-сервера. Реализуют концепцию «тонкого» клиента и «толстого» сервера. Используют традиционные языки веб-разработки: Python, PHP, Ruby, C#, Java, Go ...

Подразделяются на две категории:

- Микрофреймворки (flask)
- Традиционные фреймворки с полной функциональностью (.NET, Spring, Django)

- В некоторых языках, созданных для web разработки (PHP и др), уже встроен шаблонизатор HTML
- В отличие от таких языков, Python для веб-разработки обязательно нужны фреймворки (Django, flask и др). Для интеграции с веб-серверами в Python используются WSGI, которая основана на CGI. Например для Apache разработан модуль Apache mod_wsgi

Традиционный серверный фреймворк

- Статические файлы (статические HTML-документы, CSS, изображения, сценарии JavaScript и т.д.).
- Контроллеры (обработчики событий пользовательских действий).
- Модели (взаимодействие с БД).
- Представления (view). Шаблоны, генерирующие HTML-страницы и другое динамическое содержимое.

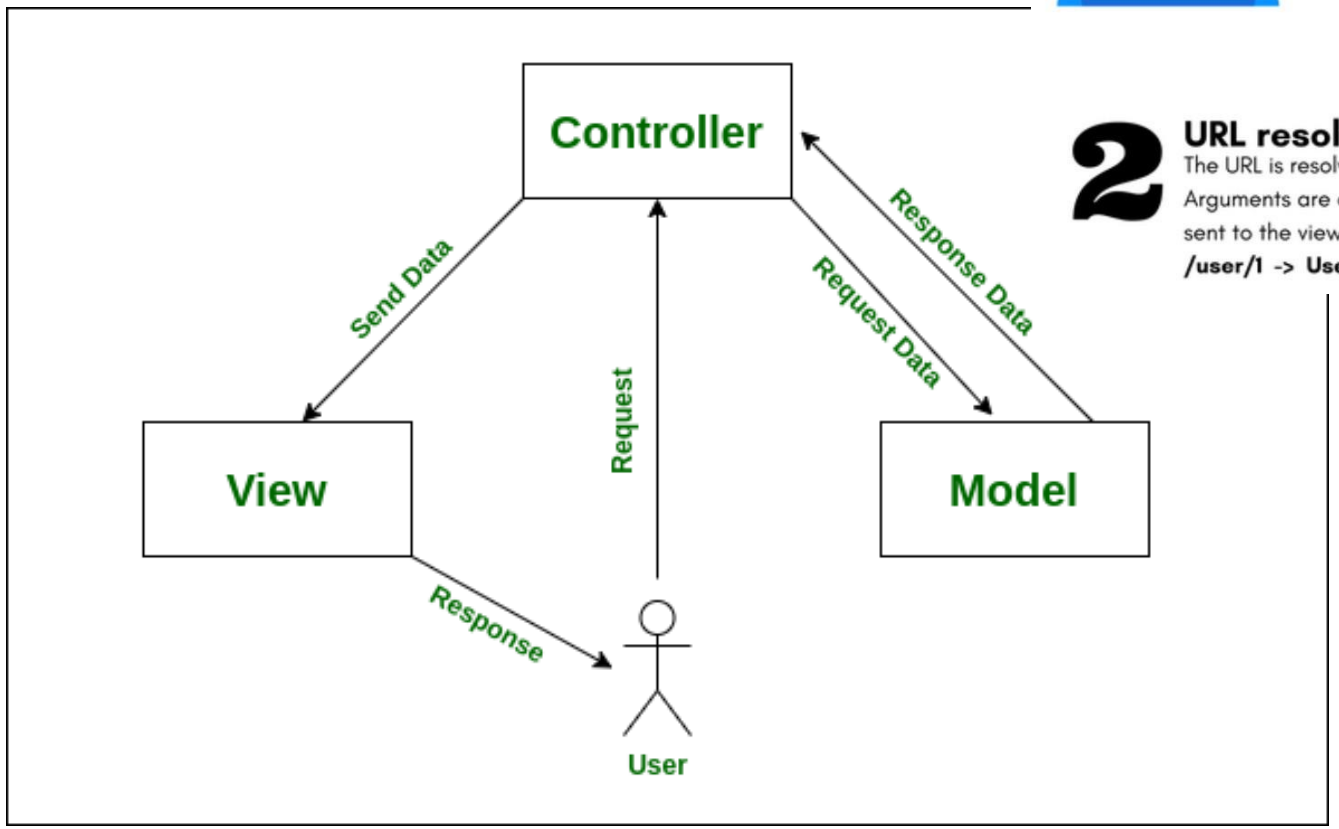
- Конфигурирование фреймворка: действия при запуске приложения, конфигурирование пользовательских сеансов (сессий), переписывание URL (привязка URL к контроллерам), безопасность (аутентификация и авторизация), кэширование, балансировка нагрузки, IOC / DI.

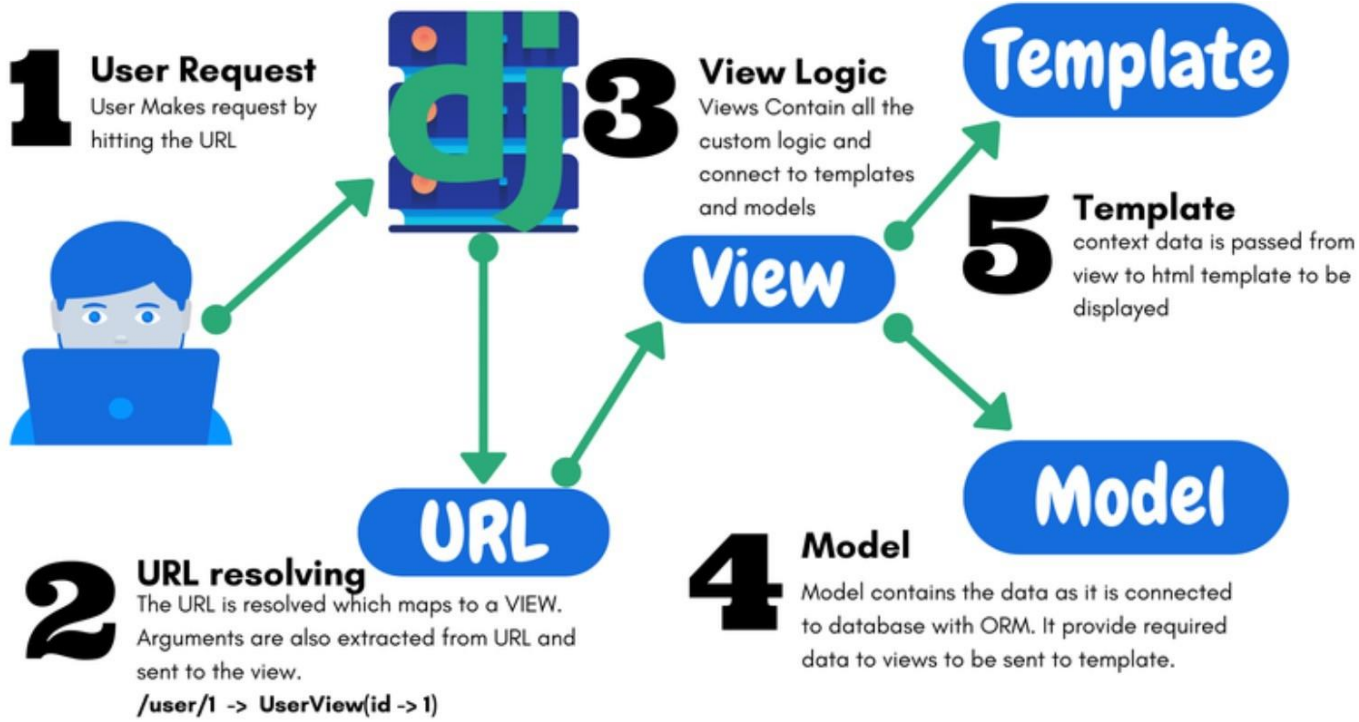
- Утилиты командной строки для управления фреймворком.

– Скаффолдинг (создание структуры проекта, генерация кода контроллеров и представлений на основе моделей, генерация кода приложения на основе специализированных описаний, генерация форм ввода и редактирования данных во время работы приложения).

– Миграции (изменение структуры базы данных на основе моделей).

MVC vs MVT

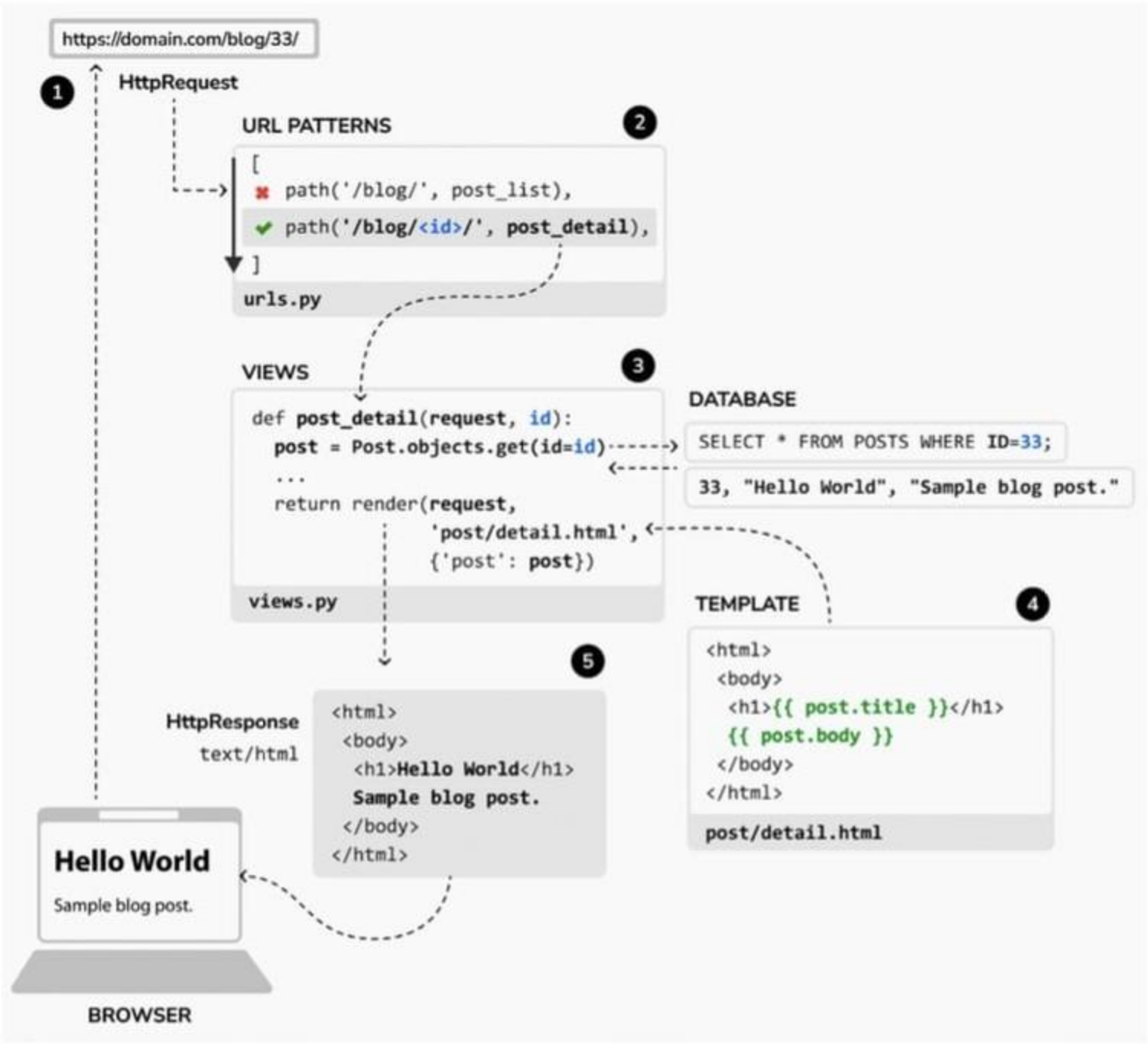




- Мы уже знаем паттерн MVC и его составляющие
- В Django используется паттерн MVT, в котором View выполняет роль Controller, а Template роль View

Django

- 1 лабораторная – это Server Side Rendering
- Django – это MVC фреймворк
- При обработке запроса сначала обрабатывается URL
- Решается, какой view будет его обрабатывать
- View обращается к БД или нейросети
- Результаты вносятся в шаблон Template, получается HTML



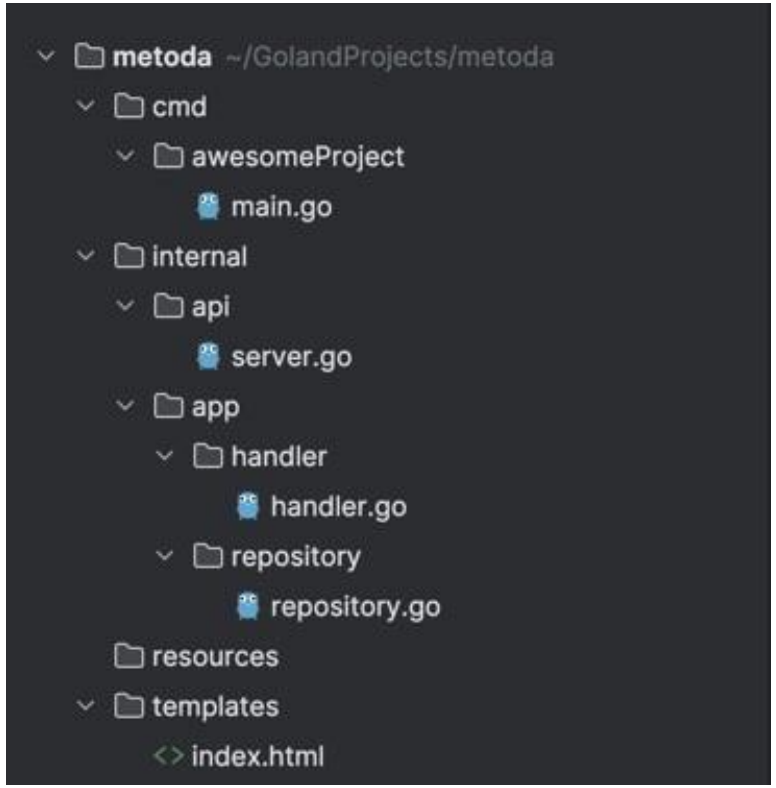
Фреймворк Django. Изучение

- Разделы документации (на русском языке)
  - https://djangodoc.ru/3.2/
  - https://django.fun/docs/django/ru/3.2/
  - https://developer.mozilla.org/ru/docs/Learn/Server-side/Django (учебник из 11 уроков)

- Важные разделы django.fun:
  - Модели (Введение в модели, запросы, миграции)
  - Представления (Обработка URL, представления на основе функций, представления на основе классов, Middleware)
  - Шаблоны (Введение, обзор языка шаблонов)
  - Формы (Введение, формы на основе моделей)
  - Администрирование

SSR проект на Golang

- На Golang у нас нет такого богатого web фреймворка, но все остается похожим: обработчики, шаблонизатор
- Обязательно сами формируем структуру проекта!




```go
func (h *Handler) GetOrder(ctx *gin.Context) {
    idStr := ctx.Param("id") // получаем id заказа из
    // через двоеточие мы указываем параметры, которые
    id, err := strconv.Atoi(idStr) // так как функция
    if err != nil {
        logrus.Error(err)
    }

    order, err := h.Repository.GetOrder(id)
    if err != nil {
        logrus.Error(err)
    }

    ctx.HTML(http.StatusOK, "order.html", gin.H{
        "order": order,
    })
}
```

Шаблонизация и коллекция

- В Go нет классов, но есть структуры
- В этом примере модель-массив услуг используется в нескольких страницах-шаблонах

```
<html lang="en">
<header>
    <h1>
        <a href="/hello">Список</a>
<!--Простой хедер, чтобы возвращаться на главную страницу-
    </h1>
</header>
    <h1>
        {{ .time }}
    </h1>
    <ul>
        {{ range .orders }}
        <li>
            <a href="/order/{{ .ID }}"> {{ .Title }} </a>
        </li>
        {{ end }}
    </ul>
</html>
```

```
type Order struct { // вот наша новая структура
  ID    int // поля структур, которые передаются в ша
  Title string // ОБЯЗАТЕЛЬНО должны быть написаны с
}

func (r *Repository) GetOrders() ([]Order, error) {
  // имитируем работу с БД. Типа мы выполнили sql зап
  orders := []Order{ // массив элементов из наших стр
    {
      ID:    1,
      Title: "first order",
    },
    {
      ID:    2,
      Title: "second order",
    },
    {
      ID:    3,
      Title: "third order",
    },
  }
  // обязательно проверяем ошибки, и если они появили
  // тут я снова искусственно обработаю "ошибку" чист
  if len(orders) == 0 {
    return nil, fmt.Errorf("массив пустой")
  }

  return orders, nil
}
```

Minio

- Вам потребуется установить S3 хранилище Minio для ваших изображений (через Docker)
- Сейчас вы **вручную добавляете** изображения и используете их в своем приложении
- Позже в ЛР-3 ваше приложение будет само загружать изображения в S3