# Dash

## Основная идея и для чего создан

Как я понял фреймворк сделан для тех людей, которые не хотят сильно запариваться с веб разработкой а именно написанием JS + HTML + CSS. Фреймворк представляет удобные классы и методы для того, чтобы разработчик сосредоточился именно на визуализации своих данных и интерактивности представления, а не на создании интерфейсов и налаживания внутренней логики сайта.

## На чем основан

Фреймворк работает под управлением flask. Внешний интерфейс отображает компоненты с помощью React.

Компоненты фреймворка представляют собой классы Python, которые копируют свойства react компонента и сериализуются как Json.

Как я понял, в dash есть некоторые инструменты, которые пакетируют компоненты React в качестве компонентов для dash. Эти инструменты используют динамическое программирование для генерации python классов, причем классы генерятся с автоматической проверкой аргументов и записей документов. Весь набор html тегов так же доступны из React.

## Что может быть сервером и как развертывать?

В документации сказано что развернуть свое приложение можно точно так же как и обычное приложение Flask. То есть можно просто посмотреть документацию Flack по развертыванию, по идее, шаги те же самые.  
Вот тут подробнее написано <https://flask.palletsprojects.com/en/1.1.x/deploying/>.

## Из возможностей

* Как я понял можно полностью кастомизировать свои элементы и все html теги доступны в Python контексте
* Приложение может быть не только одностраничным
* Диаграммы отображаются как в виде SWG так и через WebGL
* Как я понял ajax запросы можно организовать через создание своего собственного компонента React
* Есть обширная библиотека классов для большого кол-ва визуальных компонентов, которая хранится в
* Есть поддержка hot-reloading - сервер перезагружается после добавления изменений.
* Для предоставления какой то текстовой информации есть специальные классы типо Markdown.
* Довольно просто сделать приложение интерактивным, для этого используются декораторы

@app.callback( Output(component\_id='my-output', component\_property='children'), [Input(component\_id='my-input', component\_property='value')] )

def update\_output\_div(input\_value): return 'Output: {}'.format(input\_value)  
функция update\_output\_div будет вызываться при каждом изменении значения value у "#my-input".

В качестве аргументов у декоратора передается два списка - output и input, то есть может быть несколько входов для изменения параметров и выходов.

* Есть более 35 типов графиков которые представляет plotly.py

Они так же являются интерактивными и могут обновляться с помощью callback. Так же они предоставляют 4 своих собственных сигнала hoverData, clickData, selectedData, relayoutData

* Есть 3 способа шарить данные между коллбеками:
  + У пользователя, в браузере с помощью скрытого элемента, на его изменение подвязывается коллбек, и аргумент функции шарится между другими коллбеками.
  + На диске, в бд или файле
  + В shared memory space, таком как Redis.
* Есть поддержка многостраничных приложений.

Подробнее тут <https://dash.plotly.com/urls>

* Есть поддержка Live Updating Components   
  Подробнее тут <https://dash.plotly.com/live-updates>

## Производительность

## В документации сказано что проблемы с производительностью чаще всего связаны с долгой обработкой callback'ов, поэтому надо стараться делать их быстрее.Для того чтобы они работали быстрее можно воспользоваться например меморизацией и сохранять результат функции если вызов callback'а призошел с теми же аргументами.