Case study: Create mobile backend for existing Django project (Draft 1)

Defining the scope: Provide mobile application with access to the data.

Solution overview: Extend current code base by creating an API.

- Solution assumes that the objective is to extend current functionality to mobile app. Web-app already provides most of the core features we need
- Users interact with the application primarily via the web-based app. A different solution might be better if app usage is primarily.

Background

Brief description of current model of the Django website:

Web app displays **server-side rendered** pages in response to requests as well as **AJAX** calls to the backend. AJAX calls allow for page updates without reloading a whole page. Most of the **logic** happens in the server. Let's unpack that.

1. Server-side rendering – Server converts the data into HTML format and sends the complete HTML page to the browser.
2. Ajax – If the app needs to update certain parts of the page without reloading the whole page
    a. It uses AJAX to request some data from the server
    b. when the data is received on the client side, JavaScript will update the page
3. Everything else is handled by Django including:
    a. Data access layer -**model** uses object relational mapper (**ORM**) to interact with database**.** A model is a representation of your data, think of it as an interface that allows access to the underlying database without having to deal with the specifics. You can use the same model for different types of databases.
    b. Presentation layer – **template renderer.** How something should be displayed on a web page. Communication with the database through the ORM (Data-access layer)
    c. Business logic layer -**view controller.** Bridge between the template and the models. This is where you would decide for example what property data to extract from the model and which template to use to display it. Or vice-versa use search parameters from a form to obtain saved search data from the model.

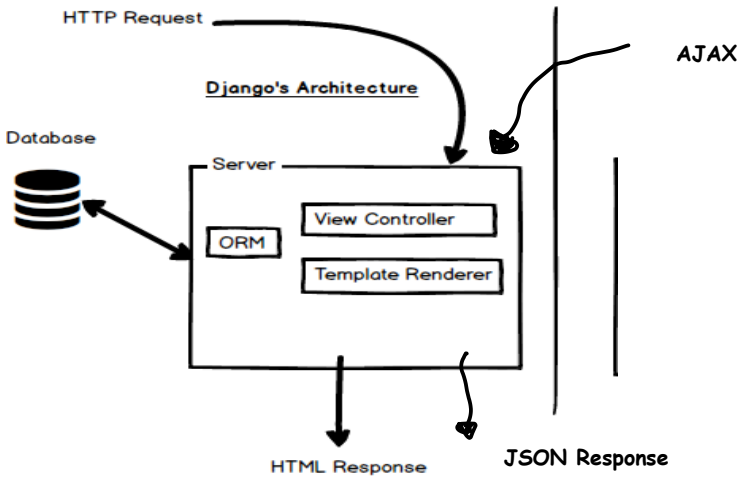

Image source

Contact: themba@teamcoda.com

Some key points to note from the above

- We do not access the database directly. We use Django's models to represent our data and let the ORM interact with the database.
- The app, using the template renderer provides the browser with fully rendered html pages. This will be a key difference to the solution below which essentially replaces the template renderer on the server side with a renderer on the application itself.

How to allow an app to access information on the database

Solution – Create an API on top of the existing Django application. The API will expose resources/data via specific end points or URIs. This will allow the BP.com app to access the information from the bedrijfspand.com Django app.

- Client/server model with app as the client requesting resources from the app via http[s].
- API will define the interface between the client and server including
    - URI – unique identifier for the resources /services available
    - HTTP methods allowable for that resource (GET,POST,PUT,DELETE)

Example of what this would look like:

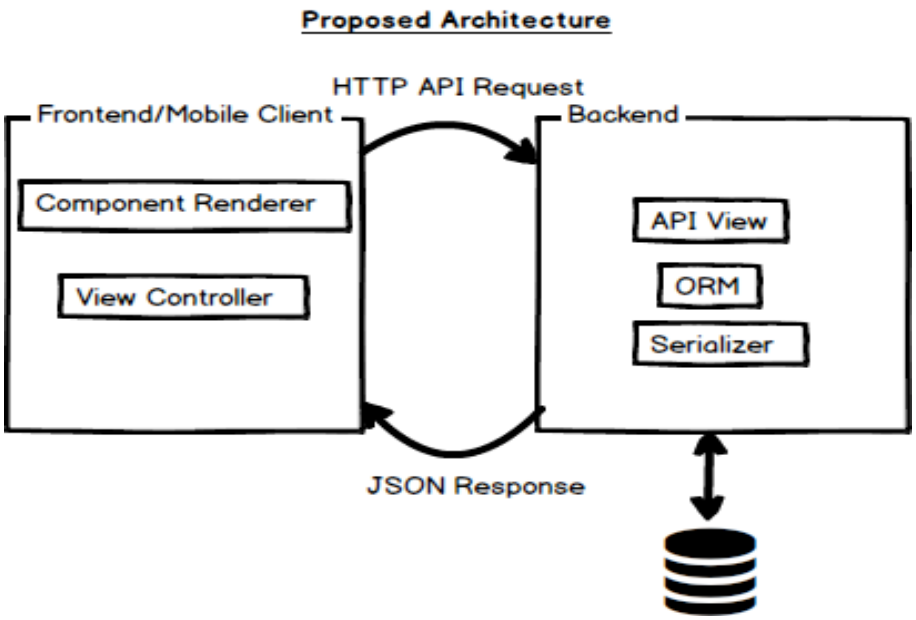

image source

Step 1: Client – makes an HTTP request to the server

- Client can be Android /IOS/ mobile web app.
- Request can be for example to authenticate a user or to provide information in response to search parameters about a property

Step 2: Server process the requests and responds with JSON response. Processing the request includes

- Authentication i.e. checking the user is valid

Contact: themba@teamcoda.com

- Authorization – check if the user has permission to access resource
- Communicating with the database and retrieving/adding/removing data
- Providing the client with the response in JSON format

Step 3: Client (mobile app) receives the information in JSON format and displays it appropriately on the application.

Tools

Custom API – Use basic http requests/responses.

- Advantages – Lightweight, easiest to implement. Slightly faster than using a framework like Django-rest-framework / Tastypie
- Disadvantages – Error prone, not in accordance with web standards, will have to recreate a lot already available code. Not recommend.

Use an available framework. e.g. Django rest framework. (Recommended)

- Advantages – well tested and documented, conforms to web standards, a lot of the basic functionality required to build a good API is built-in or can easily be added e.g. authentication, pagination etc, fits well into the Django architecture
- Disadvantages – slightly slower than a custom API using plain http requests, learning curve for new developers

Other considerations

- Scalability – whether the server is able to handle added load if mobile app usage increases
- Frontend design decisions – what tech stack to use, targeted platform

Contact: themba@teamcoda.com