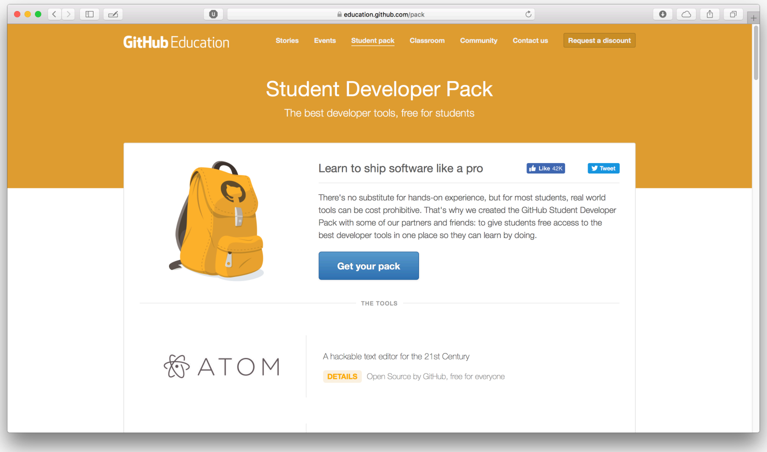
Back-end architecture

Tiberiu Vilcu
Prepared for EECS 411
Sugih Jamin
2 January 2018



Student Developer Pack

The best developer tools, free for students

Learn to ship software like a pro

There's no substitute for hands-on experience, but for most students, real world tools can be cost prohibitive. That's why we created the GitHub Student Developer Pack with some of our partners and friends: to give students free access to the best developer tools in one place so they can learn by doing.

A hackable text editor for the 21st Century

Open Source by GitHub, free for everyone

https://education.github.com/pack

Outline

1. HTTP and useful web tools
2. Designing APIs
3. Back-end services
4. Testing APIs

HTTP

(HyperText Transfer Protocol)

Request and response protocol used over TCP to serve the internet we know.

Used by almost all APIs and custom SDKs you'll encounter.

Contains header data and a payload body data (text, JSON, XML…)

HTTP requests

- GET: retrieves information from the server using the URL
- HEAD: retrieves status and header information from the server only
- POST: sends new data to the server
- PUT: updates (replaces) target data stored on the server with new data
- DELETE: removes target data stored on the server

HTTP status codes

100s: informational (nothing for you)

200s: success codes (action requested was performed by server)

300s: redirection (client takes additional action to complete)

400s: client error (the client tries actions that are incorrect or not allowed)

500s: server errors (the internal server cannot respond or execution halts somewhere)

Web API

(Application Programming Interface)

Provides the high-level connection between your back-end and your front-end mobile apps.

API endpoints are URLs where clients take action on the server.

REST API

(Representational State Transfer)

A way of allowing clients to ask for and manipulate web resources.

Response data usually sent as HTML, JSON, or XML.

Stateless and standard operations for fast performance.

API documentation

It’s important your team documents your API first before any work is done.

All endpoints should be thought of and have its contents described:

- URL
- Description of functionality
- Expected input parameters
- Expected response
- Possible error codes and reasoning

API documentation format

GET Photos (/photos)

Parameters:

- URL parameters:
  - parameter (type, default value)
- data parameters:
  - optional parameter (type, default value)

Return format:

- value (type, description)

Errors:

- error code (reasoning)

Example request with call and response

URL parameters

`http://website.com/path`**`?key=val&key2=val`** are URL parameters

The server retrieves the (key, value) parameters when processing that specific URL request to modify its response.

They should not be used to pass large or sensitive data in POST requests.

HTTPS protocol

HTTPS uses HTTP over TLS; encrypted data sent to/from server

- Need Certificate Authority to verify website to use HTTPS
- Server must support HTTPS over a different port

Let’s Encrypt

JSON

(JavaScript Object Notation)

Light data interchange format using (key, value) objects and arrays.

Text format, human-readable and language independent.

Many languages natively support JSON I/O, or if not have third-party tools.

Packing JSON

Commonly used in REST APIs to send/receive data.

Included in HTTP request bodies.

Data structures attached to request in JSON format by any language.

Dictionaries or hash maps typically used to represent data in code.

JSON example and useful tool

```
{
    "users": [{
            "username": "testuser1",
            "email": "test1@user.com",
            "password": "fakepass"
        },
        {
            "username": "testuser2",
            "email": "test2@user.com",
            "password": "fakepass"
        }
    ]
}
```

Linter: https://jsonlint.com

Domain names

Custom domain names are paid for in subscription service.

1. Purchase domain name

2. Set-up domain name on server (follow their instructions)

3. Configure the DNS provider to point to the DNS target (your app)

https://www.namecheap.com

https://domains.google/#/

https://www.godaddy.com

Basic security practices

- Permission URLs differently for different users
- Verify/sanitize all form and data inputs
- Use (third-party) authentication services for logins
- Don’t store or transmit important information unencrypted
- Don’t write own crypto code; use others’
- If using passwords, salt and hash

Basic stability practices

Don’t want 500 INTERNAL SERVER ERROR

- Validate URL/JSON parameters and values
- Validate entries in database before/after adding
- Return correct error codes for all caught exceptions

DigitalOcean

A simple, inexpensive cloud computing service.

- Deploys “Droplets” to run machines
- Has various easy-to-install applications (Django, MongoDB, Node.js)
- Is not free but gives $50 credit with GitHub developer pack

https://www.digitalocean.com

DigitalOcean pros

- Most flexible back-end option, can serve all platforms
- Can also create web apps and dynamic webpages
- Little work necessary for bare minimum product
- Simple to code, in Python (for our example)
- Custom API generally easiest to use among team
- You have your own machine to do whatever work necessary

DigitalOcean cons

- No visual/graphic tools; you ssh into a machine
- Requires you to work with native HTTP requests on the front end
- Does not have mobile-specific features like notifications

FireBase

Cloud platform to develop and expand app functionality.

- Support for iOS, Android, web apps
- Realtime database and cloud storage
- Analytics and advanced tracking features
- Online account management (also through Node.js)
- Supported by Google

https://firebase.google.com

Using Firebase on mobile apps

1. Register app through bundle ID or package name
2. Download configuration files from Firebase
3. Add Firebase SDK to project
4. Use their referenced API to interact with services

Firebase database

- JSON formatted data (not structured in tables)
- Allows rules for authentications and permissions
- Automatically backs up (with paid plan)
- Can select and insert items by keys and sub-keys
- Permission rules apply to the whole database (can’t give permissions to sub-keys or portions)

Firebase pros

- Is very easy to create a project
- Has free tier, no payment needed for base features
- Easily adds Google account authentication to your app
- Enables notifications and messages
- Houses Google Analytics, AdWords and AdMob

Firebase cons

- No console control, only through web UI
- Many features are paid with no free credits
- You’re stuck with their API and SDK; no custom endpoints
- The database is limited and does not support typical SQL queries
- You may need multiple databases
- It’s not your own machine

Heroku

Cloud platform system with various products.

- Platform: managed containers with runtime environments and ecosystems
- Postgres: SQL database service
- Redis: key-value data store
- Kafka: distributed data streams (not very relevant)

https://www.heroku.com/home

Heroku pros

- Offers analogous features to Firebase and DigitalOcean, but modular
- Has a free tier for various products
- Allows you to create your own custom API endpoints
- Has traditional database storage

Using Heroku on mobile apps

Has “Sinatra” API for iOS but requires attaching SDK

Can create a REST API server on their stack with Node.js and MongoDB:

- Set up app environment
- Provision a database and connect to the server
- Create API endpoints with Node.js and Express
- Create client-side contract to interact with API

Heroku cons

- Is confusing to know which product works best
- Is harder to get started with for the less experienced
- Requires local development setup (deployment from git)
- Container-based platform doesn’t give as much freedom as an “owned” machine

Amazon Web Services

Massive cloud platform with a plethora of products.

- Product for every cloud need you could have
- Free tier for 12 months
- Advanced features not given by other services
- Support from Amazon

https://aws.amazon.com

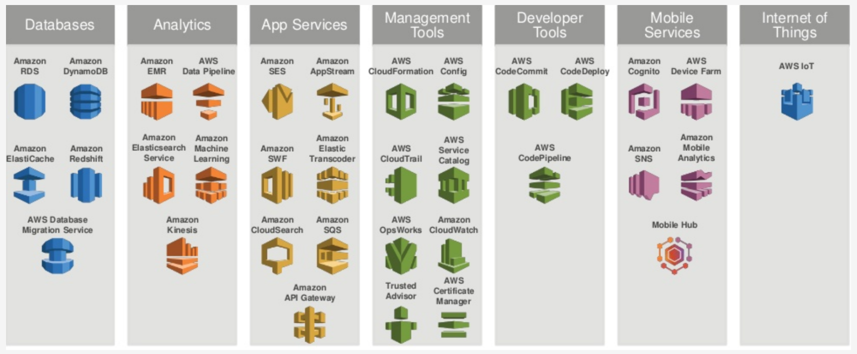
AWS pros

- Highly customizable and optimizable services
- Easy creation and destruction of various instances
- Dynamic capacities and load balancing
- Massive economies of scale
- Geographic regions and availability zones to isolate failures

AWS cons

- Easy to get lost in all of the options
- Documentation not as beginner-friendly
- Can get pricy above the free tier

AWS platform



AWS EC2

Elastic Compute Cloud: Linux or Windows machine instances.

- Easy deployment across availability zones
- Instance types for different use cases (general purpose, compute/memory/storage optimized, GPU instances)
- Scaling capacity
- Various purchasing options (on-demand, reserved, scheduled, and more)

AWS S3

Simple Storage Service: online object store.

- Natively online with HTTP access to objects
- Enables storing and retrieving any size data at any time
- Highly scalable, fast, and reliable

AWS EBS

Elastic Block Store: persistent low-level block storage.

- High low-latency performance
- Stored data automatically replicated in availability zones
- Typically a disk drive, not used online

AWS S3 vs EBS (vs EFS)

AMAZON S3	AMAZON EBS	AMAZON EFS
Can be publicly accessible	Accessible only via the given EC2 Machine	Accessible via several EC2 machines and AWS services
Web interface	File System interface	Web and file system interface
Object Storage	Block Storage	Object storage
Scalable	Hardly scalable	Scalable
Slower than EBS and EFS	Faster than S3 and EFS	Faster than S3, slower than EBS
Good for storing backups	**Is meant to be EC2 drive**	**Good for shareable applications and workloads**

AWS RDS

Relational Database Service: "traditional" database administration.

- Full capabilities of MySQL, PostgreSQL, Microsoft SQL Server, Amazon Aurora, Oracle, and more
- Resizable capacities
- Automatic failover with multiple availability zones

AWS Lightsail

Simple preconfigured setups using common developer stacks (LAMP, LEMP, MEAN, and Node.js).

Includes:

- Servers
- Storage
- DNS management and load balancing

AWS Lambda

Runs your code on-demand without thinking about servers.

You pay for the compute time consumed.

Picking a service

1. Do you need more than just a REST API?
2. Do you want to use a mobile SDK to support your backend?
3. What kind of data storage do you need?
4. Will scaling be important?
5. Do you need notifications or Google advertising?
6. What platforms/languages do you have experience with?

Testing the API

Testing should be done regularly by the back-end team.

- Test GET requests through the browser URL
- Use Postman to test POST and PUT (and other) requests
- Free app / Chrome extension to create custom requests

https://www.getpostman.com

Previous lectures

- DigitalOcean sample project repo: https://github.com/UM-EECS-441/django-project-sample-f17
- Backend development lecture slides: http://web.eecs.umich.edu/~sugih/courses/eecs441/f17/04-BackendDevelopment.pdf
- Backend development lecture recording: https://leccap.engin.umich.edu/leccap/viewer/r/9snc7M
- Up and running with AWS lecture slides: http://web.eecs.umich.edu/~sugih/courses/eecs441/f17/10-AWS.pdf
- Up and running with AWS lecture recordings: https://leccap.engin.umich.edu/leccap/viewer/r/wbvp8k https://leccap.engin.umich.edu/leccap/viewer/r/pr6Twd

Resources

- DigitalOcean: https://www.digitalocean.com
- Django setup: https://www.digitalocean.com/community/tutorials/how-to-use-the-django-one-click-install-image-for-ubuntu-16-04
- Firebase: https://firebase.google.com
- Firebase API reference: https://firebase.google.com/docs/reference/
- Heroku: https://www.heroku.com/home
- Heroku iOS tutorial: https://devcenter.heroku.com/articles/getting-started-ios-development-sinatra-cedar
- Heroku REST API tutorial: https://devcenter.heroku.com/articles/mean-apps-restful-api
- Postman: https://www.getpostman.com

Resources

- AWS EC2: https://aws.amazon.com/ec2/
- AWS S3: https://aws.amazon.com/s3/
- AWS EBS: https://aws.amazon.com/ebs/
- AWS RDS: https://aws.amazon.com/rds/
- AWS Lightsail: https://aws.amazon.com/lightsail/
- AWS Lambda: https://aws.amazon.com/lambda/