The Andela Developer Challenge
--Python Bucketlist--

The Andela Developer Challenge is founded on the premise that aspiring Technology Leaders learn programming whilst building things that matter. That the best way to learn code is by building a complete product. As we build applications that solve real problems, we internalize deeply the skills needed to be World Class Software Developers.

Contents

Introduction

App Name	**Bucketlist**
Language/Frameworks	Python/Flask/React/Postgresql
Challenge	The challenge of -----*keeping track of dreams and goals*---- is a need for -----*many individuals*----- that requires an innovative and robust solution that will ----- *allow them to remember and share the fun with others* -----.
App Description	The innovative -----*bucketlist app*----- is an application that -----*allows users to record and share things they want to achieve or experience before reaching a certain age* ----- meeting the needs of -----*keeping track of their dreams and goals* -----.
Required Features	Users create accounts Users can log in Users create, view, update and delete bucket lists. Users can add, update, view or delete items in a bucket list

Product Roadmap

To complete this project the application will need to have:

Challenge 1

1. Create a Github repository.
2. Create a folder called UI.
    a. **Without implementing the core functionality,** build the user interface with HTML/CSS . Build pages that will allow the following:
        i. User registration
        ii. User login
        iii. Creation of a bucketlist for logged in users
        iv. Viewing of one's bucketlists with UI elements for deletion and editing a bucket list as well as adding activities to a bucket list
        v. Adding items to a bucketlist

    NOTE: The interfaces are not functional at this point, just designs.

    b. Push 5 or more valid commits while building this UI.
3. Create a **UML class diagram** for your application.
4. Create a pull request and request two of your friends to review it.
5. Create a **Pivotal tracker board** mapping tasks and features required to complete challenge. This will help you keep track of your progress. It should at least capture the required features.

   Note: From here onwards, you're expected to use the Pivotal Tracker board to document and manage all implementations (chores and features).

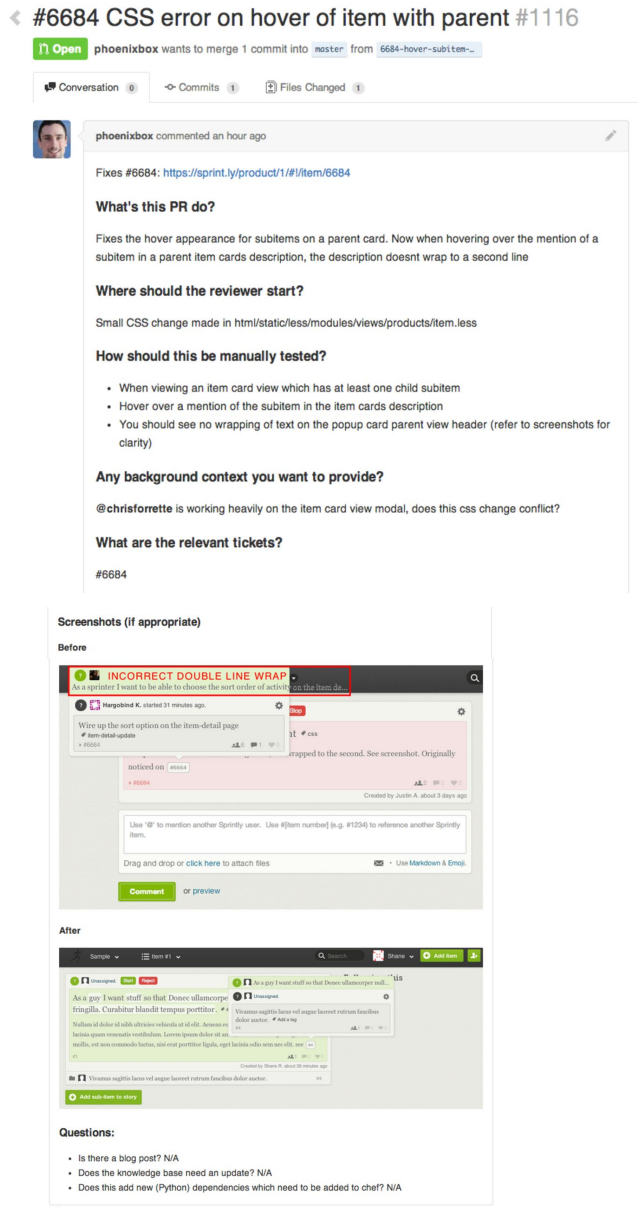
Challenge 1 Resources

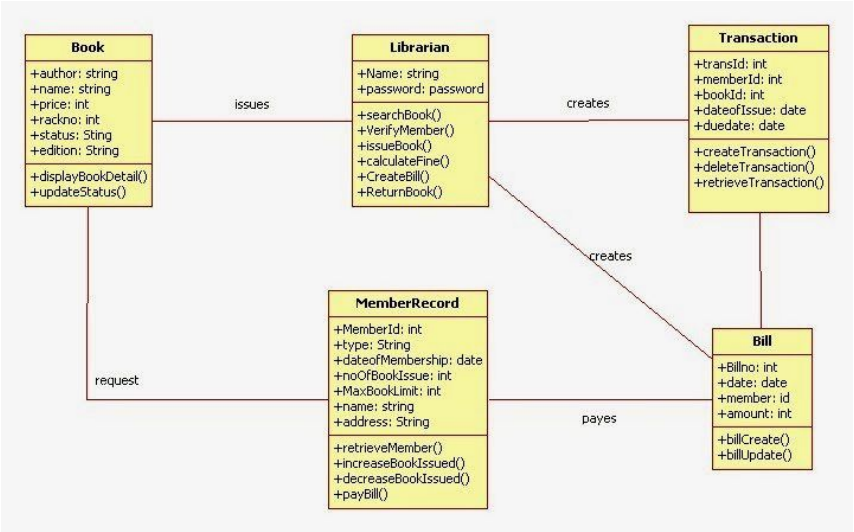
Github	Watch the Videos, version control and GIT here Use Git Workflow, Git branch, Commit Message and Pull Request (PR) standards. Also adhere to the GitHub Flow guidelines to facilitate code reviews. **- Git and Github commands tutorial** Git basics Git cheat sheet Practice Git (try.github.io) Github basics

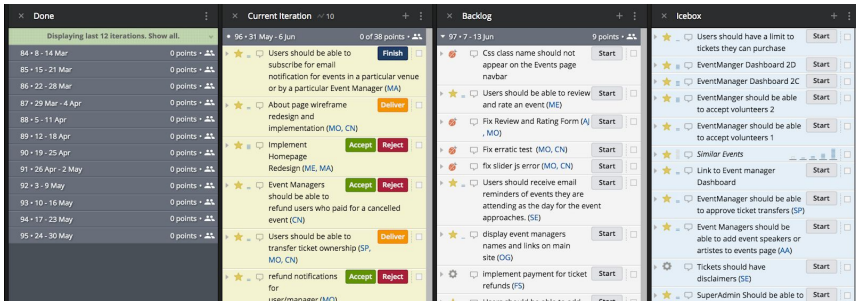
- Sample valid pull requests

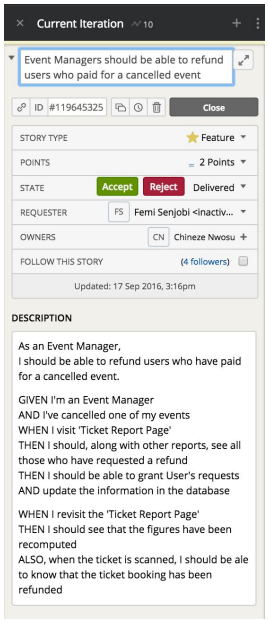
[Pull request templates]

```
#### What's this PR do?
#### Where should the reviewer start?
#### How should this be manually tested?
#### Any background context you want to provide?
#### What are the relevant tickets?
#### Screenshots (if appropriate)
#### Questions:
- Is there a blog post?
- Does the knowledge base need an update?
- Does this add new (Python) dependencies which need to be added to che
f?
```

	 #6684 CSS error on hover of item with parent #1116 Open phoenixbox wants to merge 1 commit into master from 6684-hover-subitem-… Conversation 0 · Commits 1 · Files Changed 1 phoenixbox commented an hour ago Fixes #6684: https://sprint.ly/product/1/#!/item/6684 **What's this PR do?** Fixes the hover appearance for subitems on a parent card. Now when hovering over the mention of a subitem in a parent item cards description, the description doesnt wrap to a second line **Where should the reviewer start?** Small CSS change made in html/static/less/modules/views/products/item.less **How should this be manually tested?** • When viewing an item card view which has at least one child subitem • Hover over a mention of the subitem in the item cards description • You should see no wrapping of text on the popup card parent view header (refer to screenshots for clarity) **Any background context you want to provide?** @chrisforrette is working heavily on the item card view modal, does this css change conflict? **What are the relevant tickets?** #6684 **Screenshots (if appropriate)** Before INCORRECT DOUBLE LINE WRAP After **Questions:** • Is there a blog post? N/A • Does the knowledge base need an update? N/A • Does this add new (Python) dependencies which need to be added to chef? N/A
OOP	You should have [UML Class Diagrams]() for your system. Keep the diagrams in a /designs folder. **- Sample UML models, function and class names**

<table>
<tr>
<td></td>
<td>


UML class diagram for a library management system
</td>
</tr>
<tr>
<td>HTML/CSS</td>
<td>Use SASS/SCSS to implement all custom styling</td>
</tr>
<tr>
<td>Agile</td>
<td>Use a project management tool (Pivotal Tracker) to manage your progress on the tasks. Assigns points and prioritizes tasks.
- Sample Pivotal Tracker board
 
</td>
</tr>
</table>

Challenge 1 Self-Assessment Guidelines

NOTE: Developers should use this as a general rubric to assess quality of their work. Mentors and Facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Github Commits and Version Control	Commits are not consistent at all and/or contain messages that are meaningless and not context specific.	Commits are atomic, consistent and contain meaningful messages that are context specific.	Commits follow a convention for commits using keywords based on the context of the commit like `FIXES/FEATURES/TESTS` etc.
Presence of a README	No README present.	README is present and shows information necessary to get the application running. README also show how to run tests and how to use the application. README also contains test coverage badge.	README is extremely detailed and includes more than 2 other badges and images.
HTML/CSS & FRONT END	Fails to develop HTML/CSS webpage	Successfully develops HTML/CSS webpage	Writes modular css that can be reused through

DEVELOPMENT		while observing standards such as doctype declaration, proper document structure and has consistency in the markup	markup selectors such as class, id and , Can confidently re-arrange divs
UI/UX	Solution is poorly styled and does not follow UI/UX style guides and conventions.	Solution makes good use of UI/UX conventions.	
PIVOTAL TRACKER	Does not break down the project into manageable tasks	Breaks down project into incremental stories and puts them into the appropriate categories e.g. icebox, backlog...	Updates the pivotal tracker with time/effort estimates for each assigned task, updates story cards with activity taken
OOP	Has no UML diagram	UML diagram captures attributes and methods	

Challenge 2

6. Ensure Challenge 1 is complete and merge with master.
7. Setup **Flask**.
8. Setup **Pylint** for linting and ensure your work follows PEP8 style guide requirements.
9. Setup **unit testing** libraries and ensure minimal tests written.
   Note: From here onwards, you're expected to practise Test-Driven Development.
10. Develop **features** enabling user to Create, Read, Update and Delete **non-persistent data** (no database required, data is lost when application stops) using OOP.
11. Building on the UI templates developed in Challenge 1, create a user interface to access functionality above implemented with HTML/CSS/Bootstrap or Materialize.
12. Integrate **TravisCI** for Continuous Integration in your repository (with *ReadMe* badge).
13. Integrate **test coverage reporting** (e.g. **Coveralls**) with badge in the *ReadMe.*
14. Obtain **CI badges** (e.g. from **Code Climate** and **Coveralls**) and add to *ReadMe*.
15. Create a pull request and request two of your friends to review it.
16. **Deploy** your Flask application to Heroku.

Challenge 2 Resources

Python 3.6 - Flask	What's new in Python 3.6? Create a virtual environment for your project

	Create a requirements.txt file to store your dependencies Setup Flask
Testing	Setup libraries for testing and test runners. Practise Test Driven Development.
PEP8 - Python style guides	Setup linting to ensure your code follows PEP8 standards. See guidelines for python code.
Continuous Integration	Integrate **HoundCI** for style checking commits in your PRs Integrate a CI tool (e.g. **TravisCI** or **CircleCI**) to also run tests and report pass/fail state with badge in readme and also test coverage reporting(e.g. coveralls)with badge in the readme. Obtain CI badges from **Code Climate** and **Coveralls**. *p.s this should be in the readme*
Continuous Deployment	Host a working version of the application on **Heroku.**

Challenge 2 Self-Assessment Guidelines

NOTE: Developers should use this as a general rubric to assess quality of their work. Mentors and Facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Data Structures	Fails to implement CRUD or Implements CRUD with persistence	Implements CRUD without persistence	Uses the most optimal data structure for each operation
Programming Logic	Fails to write a function that returns a value	Translates requirements into working functions which are implemented with best practices in mind	Optimizes code to effectively use system resources
Test-Driven Development	Solution did not attempt to use TDD	Writes tests that pass and achieves 70% test coverage	Writes tests that pass and achieves > 100% code coverage

OOP	Fails to write classes that incorporate basics of OOP	Writes classes with attributes and methods	Reuses code via class inheritance

Challenge 3

18. Create a new Repo in which you will develop a Flask API.
    Note: From here onwards, you're expected to practise Test-Driven Development.
19. Create models for the data which the API will be manipulating using **SQLAlchemy**.
20. Implement data persistence using **Postgresql**
21. Create a **RESTful API** using Flask with Endpoints that:
    a. Enable users to create accounts and login into the application

EndPoint	**Public Access**
POST /auth/register	TRUE
POST /auth/login	TRUE
POST /auth/logout	TRUE
POST /auth/reset-password	TRUE

    b. Enable users to create, update, view and delete a bucket list

EndPoint	**Public Access**
POST /bucketlists/	FALSE
GET /bucketlists/	FALSE
GET /bucketlists/<id>	FALSE
PUT /bucketlists/<id>	FALSE
DELETE /bucketlists/<id>	FALSE

    c. Add, update, view or delete items in a bucket list

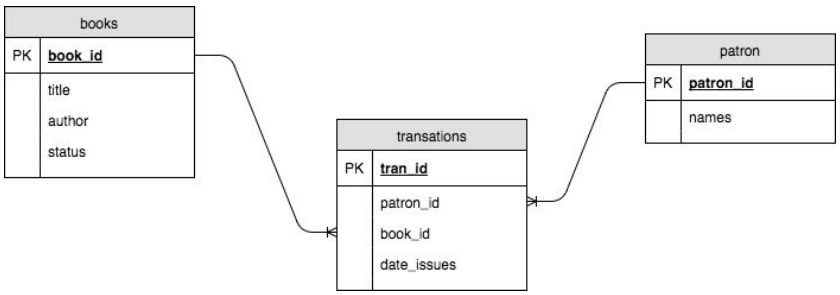
EndPoint	**Public Access**
POST /bucketlists/<id>/items/	FALSE

PUT /bucketlists/<id>/items/<item_id>	FALSE
DELETE /bucketlists/<id>/items/<item_id>	FALSE

22. Implement **Token Based Authentication** for the API such that methods besides login and register are not accessible to unauthenticated users.
23. Create scripts for handling migration of data when the data model changes.
24. Implement searching based on the name using a GET parameter **q**.
25. Implement pagination on your API so users can specify the number of results they would like to have via a GET parameter **limit**.
26. Test API with **Postman**.
27. Using a tool like Swagger or Apiary, document your API. The documentation should be accessible via your application's URL.
28. Integrate **TravisCI** for Continuous Integration in your repository (with *ReadMe* badge).
29. Integrate **test coverage reporting** (e.g. **Coveralls**) with badge in the *ReadMe*.
30. Create a pull request and request two of your friends to review it.
31. Merge with master
32. Deploy to Heroku.

Challenge 3 Resources

Databases	Postgresql **- Setting up Database** PostgreSQL installation guide- Windows PostgreSQL installation on Ubuntu
SQLAlchemy	For this task you will be creating the models for the data which your application will be manipulating. This should be done using SQLAlchemy.
Build API	Download and install the Google Chrome app **Postman.** This would be used to test the API you are building. Before you begin this section, ensure to review this material Best Practices for a pragmatic RESTful API In this task you are required to create the API endpoints described above using any of Flask, Flask-RESTful or Flask-RESTless as primary framework.
Document API	Use a tool like Swagger or Apiary to document your API **- Sample API Base URL, API Key and API Documentation** **Base URL: https://newsapi.org/v1** **API Key: 213327409d384371851777e7c7f78dfe** **Documentation: https://newsapi.org/#documentation**

Models	**- Sample data models**  Simple ERD showing that a patron can have many transactions but one transaction can only belong to one patron

Challenge 3 Self-Assessment Guidelines

NOTE: Developers should use this as a general rubric to assess quality of their work. Mentors and Facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Databases	Unable to create database models for the given project	Has a database design that is normalized and can store, update and query records from the database	Creates table relationships
Security	Fails to implement authentication and authorization in given project	Successfully implements authentication and authorization in the project	Develops features to reset passwords, delete users and creates custom and descriptive error messages
Token-Based Authentication	Does not use Token-Based authentication	Makes appropriate use of Token-Based authentication and secures all private endpoints.	
API	Fails to develop an API that meets the requirements specified	Successfully develops an api that gives access to all the specified end points	Handles a wide array of HTTP error codes and the error messages are specific

Challenge 4

33. Ensure challenge 3 is complete and merge with master

34. Create a new Repo in which you will develop a ReactJS client to consume the API you built in challenge 3.
35. Improve your front-end prototype built in challenge 1 using **Bootstrap** or **Material Design Framework**.
36. Setup **eslint** for linting and ensure you have the style guide rules configured properly.
37. Set Up a react application using create-react-app. (All Javascript **MUST** be written in **>=ES6** and should use **Babel** to transpile down to **ES5**).
    a. This should contain all the features pre-designed in previous challenges.
    b. The implementation should make use of the API built in Challenge 3.
38. Write tests for all components using Enzyme, Jest or any relevant testing utility.
39. Write End-to-End tests for all features implemented using Protractor, Nightwatch or any Selenium-based libraries.
40. Integrate **TravisCI** for Continuous Integration in your repository (with *ReadMe* badge).
41. Integrate **test coverage reporting** (e.g. **Coveralls**) with badge in the *ReadMe*.
42. Obtain **CI badges** from **Code Climate** and **Coveralls**. These should be in the *ReadMe*.
43. Integrate **HoundCI** for style checking commits in your PRs according to the ESLint configuration.
44. Ensure your front-end is also hosted on Heroku.
45. Create a pull request and request two of your friends to review it.

Challenge 4 Resources

ReactJS - Flux/Redux - Webpack - Task-runners	Use **ReactJS** with the **Flux architecture** for your implementation (see resource here) Install and configure ***Webpack*** to run mundane tasks like convert SCSS -> CSS, run your tests(Integration and unit). A task runner should be setup to handle the various tasks that the application requires which include **serving the app, and testing the app**
ES6 + Babel	All Javascript **MUST** be written in **>=ES6** and should use **Babel** to transpile down to **ES5**
OOP + SRP	Classes/modules **MUST** respect the ***SRP*** (Single Responsibility Principle) and **MUST** use the ***>=ES6*** methods of *module* ***imports and exports.***
Consume API	Find Base URL, API Key and Documentation for the API you intend your application to consume.

<u>All Resources and Assessment Guidelines</u>

Self-Assessment Guidelines

NOTE: Developers should use this as a general rubric to assess quality of their work. Mentors and Facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Presence of a README	No README present.	README is present and shows information necessary to get the application running. README also show how to run tests and how to use the application. README also contains test coverage badge.	README is extremely detailed and includes more than 2 other badges and images.
File Structure	File Structure is not organized and done in a haphazard manner.	Files are properly organized in folders named based on the function. Test are	

		organized in folders based on what they are testing.	
Comments	Solution is not commented.	Solution contains adequate comments.	Solution uses doc style comments and is self documenting.
Github Commits	Commits are not consistent at all and/or contain messages that are meaningless and not context specific.	Commits are consistent and contain meaningful messages that are context specific.	Commits follow a convention for commits using keywords based on the context of the commit like `FIXES/FEATURES/TESTS` etc.
Token Based Authentication	Token-Based authentication was not used or *solution made use of persistent session data in any way*.	Solution made appropriate use of Token-Based authentication and all private endpoints are secure.	
Code Functionality	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition.
Code Readability	Code is not easily readable or is not commented. The names for variables, classes, and procedures are inconsistent and/or not meaningful. Negligence of style guides.	Code is easily readable and necessarily commented. The names for variables, classes, and procedures are consistent and/or meaningful. Style Guides are adhered to.	
Test Coverage	Solution did not attempt to use TDD.	70% test coverage.	100% test coverage or 0% test coverage like a Bawse.
UI/UX	Solution is poorly styled and does not follow UI/UX style guides and conventions.	Solution makes good use of UI/UX conventions.	
Defense	Cannot clearly articulate why they did what they did or why certain portions of their code behaves in a certain way. Does not understand	Understands exactly what they did and is able to clearly communicate that.	In addition to knowing and explaining exactly what pieces of their code works, they can also articulate how the source code of libraries

	underlying concepts/ Copied and pasted code.		used influences code behavior.

Product Requirements

Challenge 1		
Github	Create a GitHub repository and clone it locally. Use Git Workflow, Git branch, Commit Message and Pull Request (PR) standards. Also adhere to the GitHub Flow guidelines to facilitate code reviews.	[ ]
OOP	You should have UML Class Diagrams for your system. Keep the diagrams in a /designs folder.	[ ]
HTML/CSS	**Use SASS/SCSS** to implement all custom styling	[ ]
Agile	Use a project management tool **(Pivotal Tracker)** to manage your progress on the tasks. Assigns points and prioritizes tasks.	[ ]

Challenge 2		
Python 3.6 - Flask	What's new in Python 3.6? Create a virtual environment for your project Create a requirements.txt file to store your dependencies Setup Flask	[ ]
		[ ]

Testing	Setup libraries for testing and test runners. Practise Test Driven Development.	[ ]
		[ ]

.eslint - Airbnb style guides	Use a **.eslint** in your root directory of your project as your eslint configuration (in your IDE) to expose Javascript syntax errors / nitpicks. Make sure to ***extend the airbnb styleguide***.	[ ]

PEP8 - Python style guides	Setup linting to ensure your code follows PEP8 standards. See guidelines for python code.	[ ]

Continuous Integration	Integrate **HoundCI** for style checking commits in your PRs Integrate a CI tool (e.g. **TravisCI** or **CircleCI**) to also run tests and report pass/fail state with badge in readme and also test coverage reporting(e.g. coveralls)with badge in the readme. Obtain CI badges from **Code Climate** and **Coveralls**. *p.s this should be in the readme*	[ ]
Continuous Deployment	Host a working version of the application on **Heroku.**	[ ]

Challenge 3		
Databases	Use Sqlite or Postgresql	[ ]
SQLAlchemy	For this task you will be creating the models for the data which your application will be manipulating. This should be done using SQLAlchemy.	[ ]
Build API	Download and install the Google Chrome app **Postman.** This would be used to test the API you are building. Before you begin this section, ensure to review this material Best Practices for a pragmatic RESTful API In this task you are required to create the API endpoints described above using any of Flask, Flask-RESTful or Flask-RESTless as primary framework.	[ ]

Challenge 3 & 4		
ReactJS - Flux/Redux - Webpack - Task-runners	Use **ReactJS** with the **Flux architecture** for your implementation (see resource here) Install and configure ***Webpack*** to run mundane tasks like convert SCSS -> CSS, run your tests(Integration and unit).	[ ]

	A task runner should be setup to handle the various tasks that the application requires which include **serving the app, and testing the app**	
ES6 + Babel	All Javascript **MUST** be written in **>=ES6** and should use **Babel** to transpile down to **ES5**	[ ]
OOP + SRP	Classes/modules **MUST** respect the ***SRP*** (Single Responsibility Principle) and **MUST** use the ***>=ES6*** methods of *module* ***imports and exports.***	[ ]
Consume API	Find Base URL, API Key and Documentation for the API you intend your application to consume.	[ ]
Document API	Use a tool like Swagger or Apiary to document your API	[ ]

Recommended Tutorials and Samples

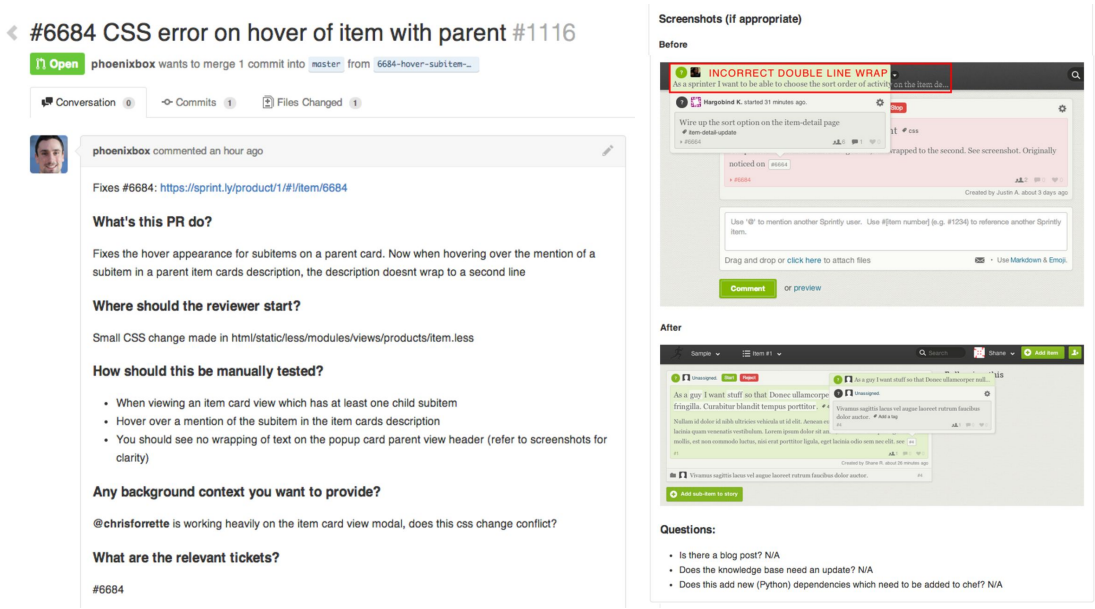
- Git and Github commands tutorial

[Git basics]
[Git cheat sheet]
[Practice Git (try.github.io)]
[Github basics]

- Sample valid pull requests

[Pull request templates]

```
#### What's this PR do?
#### Where should the reviewer start?
#### How should this be manually tested?
#### Any background context you want to provide?
#### What are the relevant tickets?
#### Screenshots (if appropriate)
#### Questions:
- Is there a blog post?
- Does the knowledge base need an update?
- Does this add new (Python) dependencies which need to be added to che
f?
```



- Flask tutorial

[The Flask Mega Tutorial]

[Getting Started with Flask]
[RealPython Flask Boilerplate]

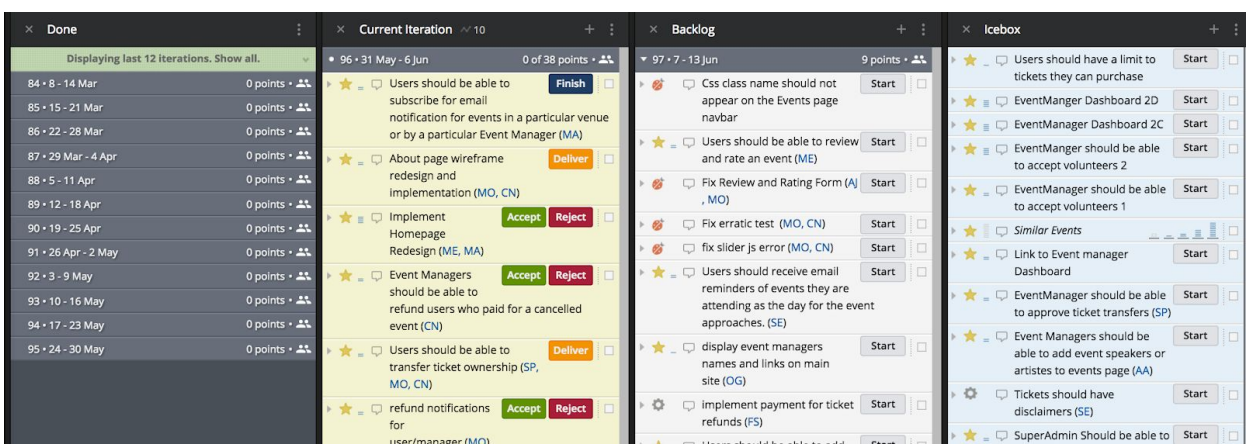
- Sample tests

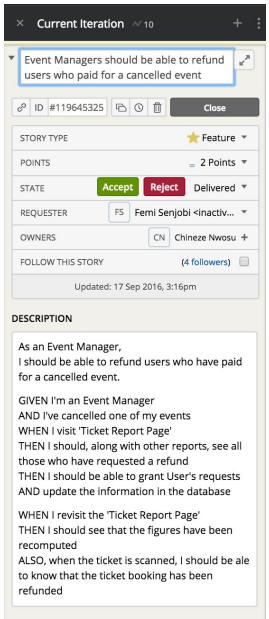
```
import unittest
from my_file import MyClass

class TestCreateRoom(unittest.TestCase):
    def test_create_room_successfully(self):
        my_class_instance = MyClass()
        initial_room_count = len(my_class_instance.all_rooms)
        blue_office = my_class.create_room("Blue", "office")
        self.assertTrue(blue_office)
        new_room_count = len(my_class_instance.all_rooms)
        self.assertEqual(new_room_count - initial_room_count, 1)
```

- TDD tutorial

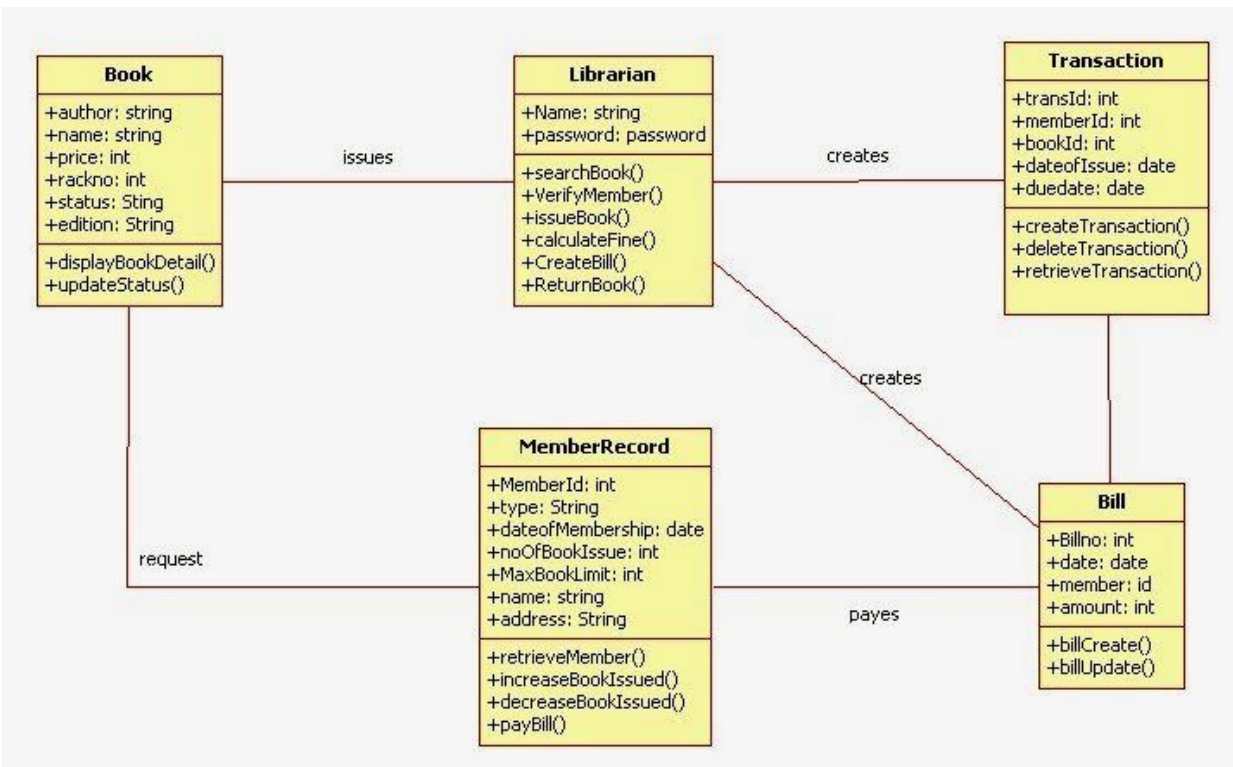
[Simple Python Unit Test Tutorial]

- Sample Pivotal Tracker board





- Sample UML models, function and class names



UML class diagram for a library management system

- Sample API Base URL, API Key and API Documentation

Base URL: https://newsapi.org/v1

API Key: 213327409d384371851777e7c7f78dfe

Documentation: https://newsapi.org/#documentation

- ReactJS

Setting up ReactJS tutorial

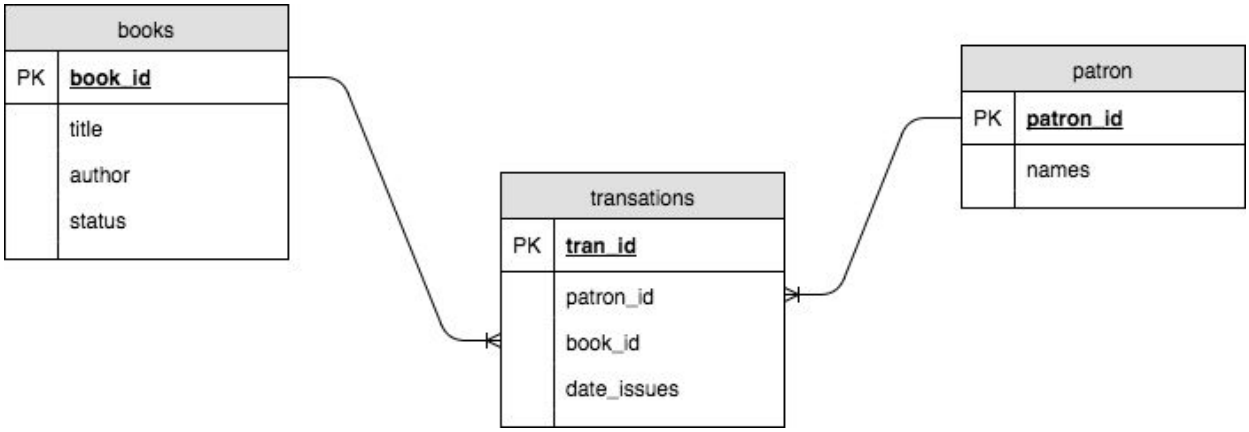
Sample ReactJS Boilerplate

- Setting up CI and Linting tutorial

TravisCI setup guide

Pylint Installation guide

- Sample data models



Simple ERD showing that a patron can have many transactions but one transaction can only belong to one patron

- Setting up Database

PostgreSQL installation guide- Windows

PostgreSQL installation on Ubuntu

- Sample API routes and endpoints

EndPoint	**Public Access**
POST /auth/login	TRUE
POST /auth/register	TRUE
POST /bucketlists/	FALSE

GET /bucketlists/	FALSE
GET /bucketlists/<id>	FALSE
PUT /bucketlists/<id>	FALSE
DELETE /bucketlists/<id>	FALSE
POST /bucketlists/<id>/items/	FALSE
PUT /bucketlists/<id>/items/<item_id>	FALSE
DELETE /bucketlists/<id>/items/<item_id>	FALSE

- API pagination

Request

GET http://localhost:5555/bucketlists?limit=20

- API searching

Request

GET http://localhost:5555/bucketlists?q=bucket1