CS 294S/W Building the Worlds Best Virtual Assistant for Mobile Devices and the Internet of Things **Spring 2017**

Homework 1

Introduction

In this homework, we will explore the Almond assistant and you will have a chance to contribute new ideas and applications to extend Thingpedia.

You are encouraged to work in the same groups as your project. You don't need to submit individually. Submissions are due April 20 electronically on the Thingpedia website. This homework is worth 5% of your grade.

To complete this assignment, you will need a phone or tablet with Android (version 5.1 or above). If do not have one, you can ask the TAs for a Android tablet to use for the assignment.

You should submit only ONE of Task 1 or Task 2

Task 0: Getting Started

This part will not be graded.

First of all, you should familiarize yourself with Thingpedia, at `https://thingpedia.stanford.edu`.

Preliminaries:

1. **All members of the group**: Register an account on Thingpedia at the above URL
2. Install Almond from the Google Play Store: `https://play.google.com/store/apps/details?id=edu.stanford.thingengine.engine`
3. **One member of the group**: apply to be a Thingpedia developer from the Settings page on the website; in the request, please indicate names and student IDs of all members of the group
4. **After the application has been approved**, link your Almond app to your Thingpedia developer account by going into Settings in the app (from the top right menu) and enabling cloud sync
5. At this point, in your app you should see the same developer key as shown in the Thingpedia website. If you filled the form right, you'll see that all members of your group have the same developer key.

When registering the Thingpedia account, please use your SUNET ID as your username. If you chose to register to Thingpedia with Facebook or Google, your username will be your email, so please change it to your SUNET ID from the Settings page in the top right.

NOTE: when registering your Thingpedia you automatically receive an account on the online version of Almond (for use with the Omlet messaging system, in iOS phones). This is what the website refers to when it talks about "Your Almond". Please disregard that (as the experience is less polished than the Android app), unless you feel adventurous.

Task 1: Writing Almond Markets

In this task you will be build a marketplace to connect buyers and sellers through Almond, similar to the Bikes market. You will define the product and its characteristics, as well as write any interesting questions and answers about the product.

The deliverable will be a working market accessible through Almond, and should be submitted in Thingpedia. Submission must happen through the "Upload new interface" button in the "Supported devices" page. For grading, the interface must include both your sunet IDs in the unique identifiers, e.g. "edu.stanford.jsmith_jdoe.fitbit". You can submit any number of times, and after you submit you can start testing the new interface. Only the last submission will be graded.

Step 1. Build your own RESTful APIs

Server You can use your own server. We also set up a server at colby.stanford.edu.

After your signed up on the first page of the class Google Spreadsheet (`http://goo.gl/tHVTs2`), you can access this server as:

```
ssh sunet-id@colby.stanford.edu
```

The password is your SUNetID, **not your SUNetID password**. You will be asked to change it immediately on the first login.

Add you own app In the following we will build a bike market as an example. First, we need to create the server app. We will use Python 3 and Django[1] as the framework to build it. Because Python is sensitive to indentation, make sure you keep the same source code formatting as the examples.

You will need to choose a *project name* and a *app name*, which must be different. Any identifier would do, but make sure it's not an existing module in the Python standard library (such as 'test' or 'mytest'). A good way to ensure that is to start with 'almond', eg. 'almonddates' or 'almondfriends'.

```
mkdir -p ~/public_html
cd ~/public_html
django-admin startproject $project_name
echo "SetHandler wsgi-script" > $project_name/$project_name/.htaccess
```

Where $project_name is the name of your project.

Next, we need to tell the app where it has been created, so it can load all other files. Add the following lines add the beginning of the file `$project_name/$project_name/wsgi.py`:

```
import sys
sys.path.append("/home/$sunet_id/public_html/$project_name")
```

Next we need to tell it where static files (CSS, JavaScript, images, etc.) are located. Modify the end of `$project_name/$project_name/settings.py` as follows:

[1] `https://docs.djangoproject.com/en/1.10/`

```
STATIC_URL = "/~$sunet_id/$project_name/static/"
STATIC_ROOT = "/home/$sunet_id/public_html/$project_name/static/"
```

(where you replaced $sunet_id and $project_name appropriately, eg. "/home/gcampagn/public_html/lol")

Then run the following commands from inside the Django project directory to collect the static files in the given location:

```
python3 manage.py collectstatic
reload-httpd
```

Next, we need to tell the app that it's fine to connect to it as 'colby.stanford.edu'. Modify the ALLOWED_HOSTS line in `$project_name/$project_name/settings.py` as:

```
ALLOWED_HOSTS = ['colby.stanford.edu']
```

Finally, we need to initialize the database, by running

```
python3 manage.py migrate
```

This will create a 'db.sqlite3' file in the project directory. You must make this file and the containing directory world-writable (`chmod a+w . db.sqlite3`) or the web server won't be able to modify it.

You can verify that your website is correctly set up by going to `https://colby.stanford.edu/~$sunet_id/$project_name/$project_name/wsgi.py/admin`. (If the URL looks weird to you because it puts a directory after a file, don't worry! That's just how Python/WSGI works.)

In this administration page you don't yet have an account. You don't need to create one, but if you want, you can use

```
python3 manage.py createsuperuser
```

to create the account.

If you change the Python files after you attempted to load the app the first time, make sure you run `reload-httpd` after you try again.

Common errors

- **Changes are not applied** Run `reload-httpd` after any change for the server to pick them up
- **Error 403 Forbidden** Make sure all files in your public_html directory are world-readable (`chmod +r -R ~/public_html`)
- **Error 500 Internal Server Error** Verify the path at the beginning of wsgi.py
- **Styling is wrong in the admin pages** Check the static files are present in the STATIC_ROOT folder, and they are loaded (eg. from Chrome Inspector or Firefox Dev Tools)
- **Syntax error after copy-pasting from the handout** Check that the quotes are regular upright quotes and not Latex quotes.

Step 1.5 The Bikes Example

Now that our website it set up, we can start creating a market inside it. You would so with the following command:

```
python3 manage.py startapp $app_name
```

$app_name is an identifier of your choice, **different from the project name**. The same restriction as the project name apply, so don't use a reserved Python name such as 'test'.

The rest of this example assumes the project name is `almondmarket` and the app name is `bikes`. You can find the full working example in `/opt/almondmarket` in colby.stanford.edu

Create models In `bikes/models.py`, define your models - essentially, your database layout, with additional metadata. In this case, we only need to create one model, `Bike`, to store the information of the bikes. Edit the `bikes/models.py` file to define all attributes about the product your selling. In our case, it looks like this:

```
from django.db import models

class Bike(models.Model):
    title = models.CharField(max_length=100)
    price = models.FloatField()
    contact = models.CharField(max_length=20)
    posted_on = models.DateTimeField(auto_now_add=True)
```

(You can find the same file as `/opt/almondmarket/bikes/models.py` in colby.stanford.edu)

Activating models To include the app, we need to add a reference to its configuration class (`BikesConfig` class in `bikes/apps.py` file) in the `INSTALLED_APPS` setting in `almondmarket/settings.py`. It'll look like this:

```
INSTALLED_APPS = [
    'bikes.apps.BikesConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Now, Django knows to include the bikes app, we can run the following two commands to apply the changes to the database.

```
python3 manage.py makemigrations bikes
python3 manage.py migrate
reload-httpd
```

You should run those 3 commands every time you change the models.py file (eg. to add or remove an attribute).

Optional step: register your model with the Admin UI If you want to use Django's builtin admin UI (for debugging), you need to append the following lines to `bikes/admin.py`:

```
from .models import Bike
admin.site.register(Bike)
```

And then `reload-httpd` as usual.

Creating a resource Now we can define the operations on the model we want to support. We're using restless for this purpose[2], and we define a *resource class*. This resource class exposes the common CRUD (create-read-update-delete) operations as HTTP methods, and maps them to a specific model.

The 'preparer' object defines how we would like the JSON data on the wire to look like. This is useful primarily to hide values that are in the database from what is returned to the user. It can be used also if, for example, the database has a user_id row mapping to a user table, but we want the JSON to contain the username (we would say something like `'username': 'user_id.username'` in that case).

Place the following code in `bikes/api.py`:

```
from restless.dj import DjangoResource
from restless.preparers import FieldsPreparer
from bikes.models import Bike

class BikeResource(DjangoResource):
    preparer = FieldsPreparer(fields={
        'id': 'id',
        'title': 'title',
        'price': 'price',
        'contact': 'contact',
        'posted_on': 'posted_on',
    })

    def is_authenticated(self):
        # Open everything wide! This is dangerous, do not do in production.
        return True

    # GET /api/ (but not hooked up yet)
    def list(self):
        return Bike.objects.all().order_by('-posted_on')[:3]

    # GET /api/<pk>/ (but not hooked up yet)
    def detail(self, pk):
        return Bike.objects.get(id=pk)
```

[2] `http://restless.readthedocs.io/en/latest/`

```
    # POST /api/ (but not hooked up yet)
    def create(self):
        return Bike.objects.create(
            title=self.data['title'],
            price=self.data['price'],
            contact=self.data['contact'],
        )

    # DELETE /api/ (but not hooked up yet)
    def delete(self, pk):
        Bike.objects.get(id=pk).delete()
```

(You can find the same file as `/opt/almondmarket/bikes/api.py` in colby.stanford.edu)

Hooking up the URLs Add your APIs to the almond market configuration in `almondmarket/urls.py`:

1. Import resource:

   ```
   from django.conf.urls import include
   from bikes.api import BikeResource
   ```

2. Add `url(r'^api/', include(BikeResource.urls()))` to the array 'urlpatterns'

As before, deploy your code by running `reload-httpd`.

Testing the APIs Run the following commands to test your APIs.

Add a new post to the database:

```
curl -X POST -H "Content-Type: application/json" -d '{
    "title": "Giant boy bike",
    "price": 100,
    "contact": "6501234567"
}' https://colby.stanford.edu/~gcampagn/almondmarket/almondmarket/wsgi.py/api/
```

Get the information of 3 latest posts

```
curl -X GET https://colby.stanford.edu/~gcampagn/almondmarket/almondmarket/wsgi.py/api/
```

Get the details of bike with id 1

```
curl -X GET https://colby.stanford.edu/~gcampagn/almondmarket/almondmarket/wsgi.py/api/1/
```

Delete a post with id 1

```
curl -X DELETE https://colby.stanford.edu/~gcampagn/almondmarket/almondmarket/wsgi.py/api/1/
```

Step 3. Write interface for Thingpedia

Write device package Put the following file in a zip file (replace bike with the name of your app). For detailed explanation of the code, see the documentation at `https://almond.stanford.edu/thingpedia/developers/thingpedia-device-intro.md`

The full code example is also available at `http://web.stanford.edu/class/cs294s/hw/almondmarket.zip`

- package.json

```
{
  "name": "edu.stanford.sunet1_sunet2.bikes",
  "version": "1.0.0",
  "description": "Interface for the Almond bike market",
  "author": "Your Group Name",
  "contributors": [
    {
      "name": "your-name",
      "email": "your-email"
    }
  ],
  "main": "device.js",
  "dependencies": {}
}
```

- device.js

```
var Tp = require('thingpedia');

module.exports = new Tp.DeviceClass({
    Name: 'AlmondMarketBikeDevice',

    _init: function _init(engine, state) {
        this.parent(engine, state);
        this.uniqueId = 'edu.stanford.sunet1_sunet2.bikes';
        this.name = "Almond Bike Market";
        this.description = "Sell and Buy 2nd hand bikes.";
    }
});
```

- post.js: channel for creating a new post (an action)

```
var Tp = require('thingpedia');
var URL = 'https://colby.stanford.edu/~gcampagn/almondmarket/almondmarket/wsgi.py/api/';
```

```
module.exports = new Tp.ChannelClass({
    Name: "PostBike",

    sendEvent: function sendEvent(event) {
        var data = JSON.stringify({
            title: event[0],
            price: event[1],
            contact: event[2]
        });
        return Tp.Helpers.Http.post(URL, data, {
            dataContentType: 'application/json',
            accept: 'application/json',
            extraHeaders: { 'Content-Length': Buffer.byteLength(data) }
        }).catch(function (error) {
            console.error('Error posting on Almond Bike Market: ' + error.message);
        });
    }
});
```

- search.js: channel for search posts (a query); this channel actually just lists all posts, and relies on ThingTalk to filter them out

```
var Tp = require('thingpedia');
var URL = 'https://colby.stanford.edu/~gcampagn/almondmarket/almondmarket/wsgi.py/api/';

module.exports = new Tp.ChannelClass({
    Name: 'SearchBikePosts',

    _init: function _init(engine, device) {
        this.parent();
        this._device = device;
        this.url = URL;
    },

    formatEvent: function formatEvent(event, filters) {
        return '%s for $%f, contact %s '.format(event[0], event[1], event[2]);
    },

    invokeQuery: function invokeQuery(filters) {
        var url = this.url;
        return Tp.Helpers.Http.get(url).then(function (data) {
            var response = JSON.parse(data);
            var posts = response.objects;
            return posts.map(function (post) {
                return [post.title, post.price, post.contact];
```

```
            });
        });
    }
});
```

- monitor.js: channel for monitoring new posts (a trigger); this repeatedly polls the list of posts from the server, compares with a local list of posts that have been already seen, and notifies for any new post

```
var Tp = require('thingpedia');
var URL = 'https://colby.stanford.edu/~gcampagn/bikes/bikes/wsgi.py/api/';

module.exports = new Tp.ChannelClass({
    Name: 'MonitorNewBikePosts',
    Extends: Tp.HttpPollingTrigger,
    RequiredCapabilities: ['channel-state'],

    _init: function _init(engine, state, device, params) {
        this.parent();
        this.state = state;
        this._device = device;
        this.interval = 10000;
        this.url = URL;
        this.postsViewed = null;
    },

    formatEvent: function formatEvent(event) {
        return 'New post found: %s for $%f, contact %s '
          .format(event[0], event[1], event[2]);
    },

    _onResponse: function _onResponse(data) {
        var _this = this;
        var response = JSON.parse(data);
        var posts = response.objects;
        this.postsViewed = this.state.get('posts-viewed');
        posts.map(function (post) {
            // This code is to prevent multiple notifications
            // for the same post
            if (_this.postsViewed === undefined ||
                _this.postsViewed.indexOf(post.id) < 0) {
                if (_this.postsViewed === undefined)
                    _this.state.set('posts-viewed', [post.id]);
                else {
                    var viewed = _this.state.get('posts-viewed');
```

```
                    viewed.push(post.id);
                    _this.state.set('posts-viewed', viewed);
                }
                _this.emitEvent([post.title, post.price, post.contact]);
            }
        });
    }
});
```

Device Metadata The metadata must be filled using the built-in tool on Thingpedia page when you publish your interface. It should not be in the zip file.

```
{
  "params": {},
  "types": [
    "data-source",
    "service"
  ],
  "child_types": [],
  "auth": {
    "type": "none"
  },
  "triggers": {
    "monitor": {
      "args": [
        {
          "name": "title",
          "type": "String",
          "question": "",
          "required": false
        },
        {
          "name": "price",
          "type": "Number",
          "question": "",
          "required": false
        },
        {
          "name": "contact",
          "type": "PhoneNumber",
          "question": "",
          "required": false
        }
      ],
      "doc": "monitor for bike posts",
```

```
    "confirmation": "monitor bike posts on almond market",
    "canonical": "monitor second hand bikes",
    "examples": [
      "monitor second hand bike posts"
    ]
  }
},
"actions": {
  "post": {
    "args": [
      {
        "name": "title",
        "type": "String",
        "question": "",
        "required": false
      },
      {
        "name": "price",
        "type": "Number",
        "question": "",
        "required": false
      },
      {
        "name": "contact",
        "type": "PhoneNumber",
        "question": "",
        "required": false
      }
    ],
    "doc": "post on almond bike market",
    "confirmation": "post  a bike with title $title for $price dollars on almond bike ma
    "canonical": "post",
    "examples": [
      "post on almond bike market",
      "post a bike for $price dollars on almond bike market"
    ]
  }
},
"queries": {
  "search": {
    "args": [
      {
        "name": "title",
        "type": "String",
        "question": "",
```

```
          "required": false
        },
        {
          "name": "price",
          "type": "Number",
          "question": "",
          "required": false
        },
        {
          "name": "contact",
          "type": "PhoneNumber",
          "question": "",
          "required": false
        }
      ],
      "doc": "search for bike posts",
      "confirmation": "retrieve bike posts on almond market",
      "canonical": "search second hand bikes",
      "examples": [
        "search second hand bikes"
      ]
    }
  },
  "global-name": "almond_bike_market"
}
```

Task 2: Writing interfaces

Your Task

In this section, your group will develop a new Thingpedia interface for everyone to use.

To obtain full credit for this task, you must complete both the configuration part of the interface and at least 2 functions (triggers or actions). Each function must have at least 3 example sentences, plus confirmation and canonical.

Logistics

Your group must sign up at `http://goo.gl/tHVTs2` indicating what interface you wish to implement (in the sheet marked "Homework 1 Task 2").

If you wish to use a physical IoT device, we have available (on a first come first served basis) a FitBit, a Bluetooth heart rate monitor, a couple of iBeacon tags, and a Raspberry Pi. To request them, sign up on the Google Spreadsheet.

Submission must happen through the "Upload new interface" button in the "Supported devices" page. For grading, the interface must include both your sunet IDs in the unique identifiers, e.g. "edu.stanford.jsmith_jdoe.fitbit". You can submit any number of times, and after you submit you can start testing the new interface. Only the last submission will be graded.

You can use the code of existing interface at starting point in writing your own. You can find it at `https://github.com/Stanford-Mobisocial-IoT-Lab/thingpedia-common-devices`, and you can ask the TA for explanations.