Today’s topic

- The homework
- Analysis of existing interfaces
The homework

- Posted on the website yesterday
- Due next Thursday (March 3rd)
- Task 1: write two apps
- Task 2: write an interface and an app
Task 0: setting up ThingEngine

- If you have not already, go do it now
- If it does not work, come to me after the session
- Go apply for developer account too!
Task 1: write two apps

- Look at the examples that are already in the collection
- Look for the supported interfaces in the dev portal
- Look at the ThingTalk documentation in the dev portal
Testing apps

My Apps

New fancy app
This app is really cool
Show code

Publish  Delete

Create a new app

My Published Apps

<table>
<thead>
<tr>
<th>Analyze love</th>
<th>How much did I sleep?</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://xkcd.com/601/">http://xkcd.com/601/</a></td>
<td>Ask Sabrina!</td>
</tr>
<tr>
<td>#jokes</td>
<td>#fitness #asksabrina</td>
</tr>
<tr>
<td>Show code</td>
<td>Show code</td>
</tr>
</tbody>
</table>
Task 2: write an app and an interface

- Choose what interface you want, and sign up for it
- Different groups => different interfaces!
The existing code

- One folder per thing, with manifests at the top
const Tp = require('thingpedia');
module.exports = new Tp.DeviceClass({
    Name: 'BodyTraceScaleDevice',
    Kinds: ['scale'],
    _init: function(engine, state) {
        this.parent(engine, state);
        this.serial = state.serial;
        this.username = state.username;
        this.password = state.password;
        this.uniqueId = 'com.bodytrace.scale-' + this.serial;
        this.name = "BodyTrace Scale " + this.serial;
        this.description = "This is a BodyTrace Scale " + "owned by " + this.username;
    },
});
Handling Auth

- Option 0: no auth at all
- Option 1: username and password
- Option 2: OAuth
The OAuth model

1. User clicks on button
2. Redirect to login page by service provider
3. User approves login
4. Redirect to our server with auth code
5. Server calls service provider to convert auth code into access token
6. Server uses access token to make requests
The OAuth model & you

1. User clicks on button
2. **Your code sets up the redirect**
3. Redirect to login page by service provider
4. User approves login
5. Redirect to our server with auth code
6. **Your code calls service provider to convert auth code into access token**
7. **Your code uses access token to make requests**
OAuth in Jawbone

UseOAuth2: Tp.Helpers.OAuth2({
    kind: 'com.jawbone.up',
    client_id: 'v3sYocgyPaE',
    client_secret: '...',
    scope: ['basic_read', 'extended_read', ...],
    authorize: 'https://jawbone.com/auth/oauth2/auth',
    get_access_token: 'https://jawbone.com/auth/oauth2/token',
    callback: // callback when OAuth is done
})
Completing Auth

callback: function(engine, accessToken, refreshToken) {
  var auth = 'Bearer ' + accessToken;
  return Tp.Helpers.Http.get(
    'https://jawbone.com/nudge/api/v.1.1/users/@me',
    { auth: auth,
      accept: 'application/json' })
    .then(function(response) {
      var parsed = JSON.parse(response);
      return engine.devices.loadOneDevice({
        kind: 'com.jawbone.up',
        accessToken: accessToken,
        refreshToken: refreshToken,
        userId: parsed.data.xid,
        userName: parsed.data.first + ' ' + parsed.data.last,
        true);
    });
}
More complex forms of Auth

- That was OAuth 2.0, aka "good OAuth"
- Twitter uses OAuth 1.0
- Significantly less standard
- Solve that by providing custom code in UseOAuth2 hook
Packaging up your device

{
    "name": "thingengine-device-bodytrace-scale",
    "version": "1.0.0",
    "description": "BodyTrace Scale support in ThingEngine",
    "author": "Stanford IoT lab",
    "main": "device.js",
    "dependencies": {
        "q": "~1.4.1"
    }
}
# Testing your device

<table>
<thead>
<tr>
<th>App Collection</th>
<th>My Sabrina</th>
<th>Developers</th>
<th>Administration</th>
<th>Settings</th>
<th>Log Out</th>
<th>About</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Sabrina
Your magic assistant, at your service.

- **Update**

## SportRadar
Watch sport results live! Includes support for the NFL, the NBA and European Soccer

- **Update**

## ThingEngine™ Test Device
A device to test ThingEngine™. It does nothing

- **Update**

## ThingTV
Your ThingTV in your living room, bedroom, office...

- **Update**

## Twitter Account
Connect your ThingEngine with Twitter

- **Update**

| Upload new interface |
The submission page

- Thing ID: as described in the homework
- Thing name and description: as shown in the dev portal
- Always check "This interface requires additional JS code"
The manifest

{
  "auth": { "type": "oauth2" },
  "types": ["online-account"],
  "global-name": "google",
  "params": {},
  "triggers": {},
  "actions": {}
}
module.exports = new Tp.ChannelClass({
    Name: 'MyChannel',
    _init: function(engine, device) {
        // constructor
    },
    _doOpen: function() {
        // start
    },
    _doClose: function() {
        // stop
    }
});
The easy part: actions

module.exports = new Tp.ChannelClass({
    Name: 'TwitterSinkChannel',
    Extends: Tp.SimpleAction,
    _init: function(engine, device) {
        this.parent();
        this._twitter = // make twitter API client
    },

    _doInvoke: function(status) {
        console.log('Posting Twitter status', status);

        this._twitter.postTweet({ status: status }, function(err) {
            console.log('Tweeting failed: ' + err);
        }, function() {
        }
    }
});
Polling triggers

```javascript
module.exports = new Tp.ChannelClass(
    {
        Name: 'RESTPollingChannel',
        interval: 3600 * 1000 * 3,
        _init: function(engine, device) {
            this.parent();
            this.url = ...;
        },
        _onResponse: function(response) {
            // handle response
        }
    },
);
```
Handling the response

```javascript
_onResponse: function(response) {
  return Q.nfcall(xml2js.parseString, response)
    .then(function(result) {
      var temp = result.weatherdata['product'][0].time[0];
      var time = new Date(temp.$.to);
      var temperature = temp.location[0].temperature[0].$.value;
      var humidity = temp.location[0].humidity[0].$.value;
      var event = [time, temperature, humidity];

      this.emitEvent(event);
    });
}
```
Other examples

- BodyTrace .source
- SportRadar .nfl, .nba_team, .soccer_eu_team
What about Jawbone?

- Jawbone has many very similar endpoints
- source.js has the common code
- Each channel instantiates source.js
Better than polling

- Set up persistent HTTP connection
- Handle items as they come
- Example: Twitter .source
Recap: what to look at

- BodyTrace: password auth, stateful polling trigger
- Sportradar: stateless polling trigger
- Jawbone: simple OAuth
- Twitter: complex OAuth, non-polling trigger
What NOT to look at

- Do not look at
  - Google, Facebook (stubs)
  - Bluetooth Generic (placeholder)
  - Heatpad (XMPP & hacky)
  - Test (mock device)
  - Weather (broken)
  - LinkedIn (is a "generic device")
Further help

- Hackathon sessions tomorrow (Saturday) and Monday
- 2pm to 5pm in Gates 409
- Come and talk to me if you’re attempting extra-credit
- Come and talk to me if you want to roll your own ThingPedia