Open-Knesset Developers Documentation

Release 0.5

Open-Knesset

Jun 08, 2017

Contents

Open Knesset's aim is making the israeli Knesset more transparent. The project is Python and Django based.

Note: This guide is for developers working on Open Knesset's code.

For 3rd party applications and services developers, please see the API documentation.

Some useful project links:

- Our code repository
- The issue tracker (bugs, tasks etc.)
- The production Open Knesset site
- The Hasadna site (our parent organization)

Important: If not done already, please subscribe to the Open Knesset Developers group.

Contents:

CHAPTER 1

GitHub and Source Code

The Open Knesset code is hosted on GitHub, and uses `git` for distributed version control.

Create a GitHub account

If not done already, goto the github's sign up, and create a user account.

Once done, login.

Forking the Open Knesset project

Forking the project creates your personal repository of the source code. Goto to the Open Knesset repository and fork the project.

CHAPTER 2

Setting up the development environment

There are several initial requirements for Open-Knesset, mainly:

- Python 2.x (including dev files)
- setuptools
- virtualenv
- pip (version 8 and up)
- git

For simple installation using http://c9.io/ cloud IDE, just clone an existing open knesset workspace: https://c9.io/orihoch/oknesset/

Once inside the worksapce, continue with the "Initial Testing, Development DB & Server" section

The section will guide you on:

Linux

Installing Initial Requirements

On Linux we'll be creating a clean virtualenv, so in addtion we'll need developer tools (to compile PIL, lxml etc).

Debian and derivatives like Ubuntu and Mint, Including c9.io

```
sudo apt-get update
sudo apt-get install build-essential git python python-dev python-setuptools python-
↪virtualenv python-pip
sudo apt-get install libjpeg-dev libfreetype6 libfreetype6-dev
sudo apt-get build-dep python-imaging
sudo apt-get build-dep python-lxml
```

Fedora

```
sudo yum groupinstall "Development Tools" "Development Libraries"
sudo yum install git python python-devel python-setuptools python-virtualenv python-
↪pip libjpeg-turbo-devel libpng-devel libxml2-devel libxslt-devel
```

Creating and Activating the virtualenv

Navigate in a terminal to the directory you want the environment created in (usually under your home directory). We'll name the created environment `oknesset`.

Once in that directory:

```
virtualenv oknesset
```

Warning: In case you have both Python 2 and 3 installed, please make sure the virtualenv is created with Python 2. If that's not the case, pass the correct python executable to the virtualenv command. e.g:

```
virtualenv -p python2 oknesset
```

To check which is the default interpreter virtualenv will use, run `virtualenv -h` and check in the output the default for *-p* flag.

We need to *activate* the virtual environment (it mainly modifies the paths so that correct packages and bin directories will be found) each time we wish to work on the code.

In Linux we'll source the activation script (to set env vars):

```
cd oknesset/
. bin/activate
```

Note the changed prompt which includes the virtualenv's name.

Getting the Source Code (a.k.a Cloning)

Now we'll clone the forked repository into the virutalenv. Make sure you're in the *oknesset* directory and run:

```
git clone https://github.com/your-username/Open-Knesset.git
```

Replace *your-username* with the username you've registered at GitHub.

Note: You can also clone with ssh keys, in that case follow the github guide on ssh keys. Once you've done that, your clone command will look like:

```
git@github.com:your-username/Open-Knesset.git
```

For c9.io you will have to use ssh keys - you will need to add c9.io ssh key to your GitHub Profile's trusted keys.

Note: If you have forked Open-Knesset in the past, make sure you have the latest version before proceeding to installation, by invoking:

```
git remote add hasadna https://github.com/hasadna/Open-Knesset.git
git pull hasadna master
git push origin master
```

Installing requirements

Still in the terminal with the virtualenv activated, inside the *oknesset* directory, run:

```
pip install --upgrade pip
pip install -r Open-Knesset/requirements.txt
```

And wait ...

Once done, proceed to *Initial Testing, Development DB & Server*.

MS Windows

Installing Initial Requirements

On MS Windows the process is more manual. We'll start by downloading and installing Python and some packages.

Important: The documentation here assumes you'll accept defaults:

- Python installed into `C:\Python27`
- Virtualenv will be created at `C:\oknesset`

If you've changed those, please adjust the instructions accordingly.

Python and packages

Python

Download the latest Python 2.7 MSI installer matching your architecture (32 or 64 bit). As of this writing, the latest one is 2.7.8.

Once downloaded, run the installer, and accept defaults.

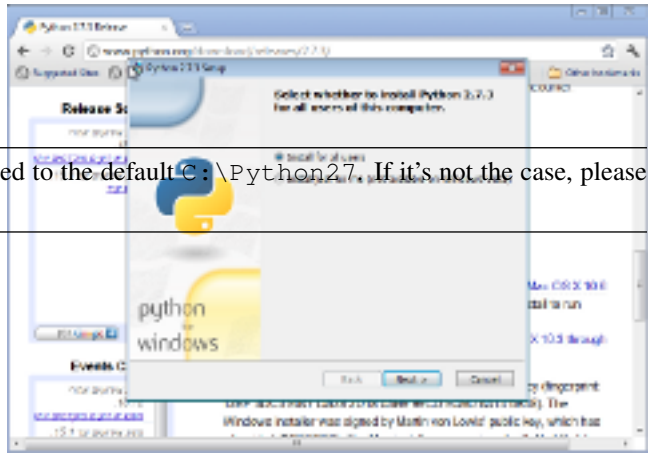


Fig. 2.1: Python 2.7 install (click to enlarge)

Important: The documentation assumes you've installed to the default `C:\Python27`. If it's not the case, please adjust accordingly.

distribute

distribute replaces setuptools and makes our windows install simpler (as setuptools for python2.7 on windows

has problems on 64bit platforms and needs a different installation method).

Download the distribute setup script and run it, either automatically via your browser, by double-clicking it in windows explorer or manually by running:

```
python␣
↪\path\to\the\download\directory\distribute_
↪setup.py
```

pip and virtualenv

We'll install them with distribute. Open a command window, and:

```
cd c:\Python27\Scripts
easy_install pip
pip install virtualenv
```

Pillow, lxml and ujson

Since compiling those packages (inside the virtualenv) is not an easy task, we'll install them separately and instruct virtualenv to use python's global site-packages (not pure, but will make things easier for MS Windows developers).

- Download and run the exe installer matching your architecture for lxml (version 2.3.x)
- Install *Pillow* with easy_install by running:

  ```
  easy_install Pillow==2.4.0
  ```

- Download and run the exe install matching you architecture for ujson.

git and GitHub tools

The Open Knesset code is hosted on GitHub, and uses `git` for distributed version control. The easiest way to install them on windows is with GitHub for Windows (download from the top right corner).

Run the installer, it'll start and download the rest of the needed packages:

Run the GitHub program (you should have an icon on the desktop), and sign in with your username and password. This should also extract git, and create a ssh key and upload the public part to GitHub.

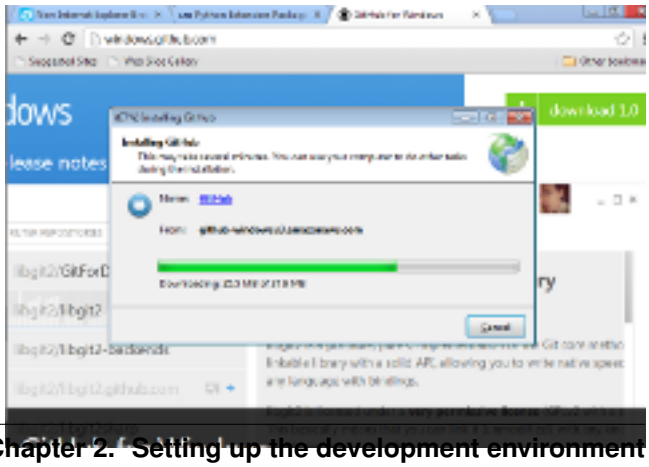


Fig. 2.2: GitHub tools install (click to enlarge)

Creating and Activating the virtualenv

From the desktop (or programs menu) run *Git Shell*

, it's a shell with git already configured, in the shell:

```
cd C:\
C:\Python27\Scripts\virtualenv␣
↪--distribute␣
↪--system-site-packages oknesset
```

Note: If this command fails:

- You probably have an older virtualenv installed. The quickest work-around is to replace it with the latest version:

  ```
  cd C:\Python27\Lib\site-packages
  del .\virtualenv*
  ..\..\Scripts\easy_install.exe virtualenv
  ```

- Another problem may be that you had PYTHONPATH environment variable configured, in that case, unset it.

We need to *activate* the virtual environment (it mainly modifies the paths so that correct *Lib* and *Scripts* directories will be found) each time we wish to work on the code.

```
cd oknesset
Scripts\activate
```

Note the changed prompt with includes the virtualenv's name.

Getting the Source Code (a.k.a Cloning)

First, we need to fork the Open-Knesset repository on github.

Now we'll clone the forked repository into the virutalenv. Make sure you're in the *oknesset* directory and run:

```
git clone git@github.com:your-name/Open-Knesset.git
```

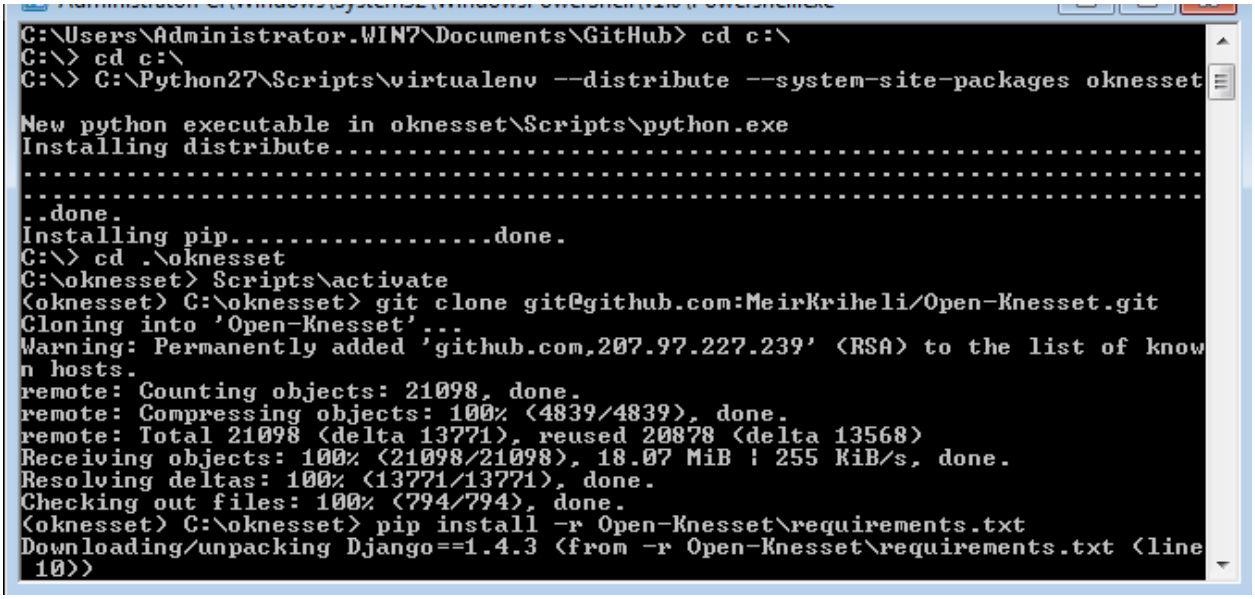
Replace *your-username* with the username you've registered at GitHub.

Installing requirements

In the Git Shell command window, with the virtualenv activated, inside the *oknesset* directory, run:

```
pip install -r Open-Knesset/requirements.txt
```

And wait ... See an example in the following screenshot:



Once done, proceed to *Initial Testing, Development DB & Server*.

OS X

Important: The info here is based on the post Fixing Python, virtualenv and pip on Mountain Lion

Command Line Tools

- Go to https://developer.apple.com/downloads
- Search for "command line tools"
- Download and install the version right for your OS

Install pip and virtualenv

```
sudo easy_install pip
sudo pip install virtualenv
```

Install basic dependencies

Install homebrew:

```
ruby -e "$(curl -fsSkL raw.github.com/mxcl/homebrew/go)"
```

Install binary python libraries build dependencies:

```
brew install jpeg libpng libxml2 libxslt
```

Add locale settings (in case you're not UTF-8), put in your `~/.profile`:

```
export LANG="en_US.UTF-8"
export LC_COLLATE="en_US.UTF-8"
export LC_CTYPE="en_US.UTF-8"
export LC_MESSAGES="en_US.UTF-8"
export LC_MONETARY="en_US.UTF-8"
export LC_NUMERIC="en_US.UTF-8"
export LC_TIME="en_US.UTF-8"
export LC_ALL=
```

And source them (to have them updated in the current shell):

```
source ~/.profile
```

Creating and Activating the virtualenv

Navigate in a terminal to the directory you want the environment created in (usually under your home directory). We'll name the created environment `oknesset`.

Once in that directory:

```
virtualenv oknesset
```

We need to *activate* the virtual environment (it mainly modifies the paths so that correct packages and bin directories will be found) each time we wish to work on the code.

To do it, we'll source the activation script (to set env vars):

```
cd oknesset/
. bin/activate
```

Note the changed prompt which includes the virtualenv's name.

Installing PDF Command Line tools (Optional. This is only needed if you want to work on the scrapers)

Now we will install Poppler, which is a package that contains *pdfinfo* and *pdftotext*. Both are requirements for running the scrapers.

```
brew install poppler
```

Note that oknesset will look for the XPATH tools on the *PATH* environment variable.

Getting the Source Code (a.k.a Cloning)

Now we'll clone the forked repository into the virutalenv. Make sure you're in the *oknesset* directory and run:

```
git clone https://github.com/your-username/Open-Knesset.git
```

Replace *your-username* with the username you've registered at GitHub.

Note: You can also clone with ssh keys, in that case follow the github guide on ssh keys. Once you've done that, your clone command will look like:

```
git@github.com:your-username/Open-Knesset.git
```

Installing requirements

Still in the terminal with the virtualenv activated, inside the *oknesset* directory, run:

```
pip install -r Open-Knesset/requirements.txt
```

And wait ...

Once done, proceed to *Initial Testing, Development DB & Server*.

Initial Testing, Development DB & Server

After you've installed the base environment, it's time to run the tests and get an initial development db.

Important:

- Linux users: you can replace `python manage.py` with `./manage.py` for less typing
- Run the manage.py commands from the *Open-Knesset* directory, with the virtualenv activated.
- If you used the c9.io worksapce, you should run the following command to get you in the write directory: cd oknesset/Open-Knesset/ && . ../bin/activate

Running Tests

```
cd Open-Knesset
python manage.py test
```

Download the Development DB

Download and extract dev.db.zip or dev.db.bz2 (bz2 is smaller). After unpacking, **place dev.db in the 'Open-Knesset' directory**.

On c9.io (or similar linux environment) you can write the following code:

```
wget http://oknesset-devdb.s3.amazonaws.com/dev.db.zip
unzip dev.db.zip
```

To make sure everything is up to date, run the database schema migrations:

```
python manage.py migrate
```

You might want to create your own superuser:

On the c9.io environment there is a superuser preconfigured: admin / 123456

```
python manage.py createsuperuser
```

Running the Development server

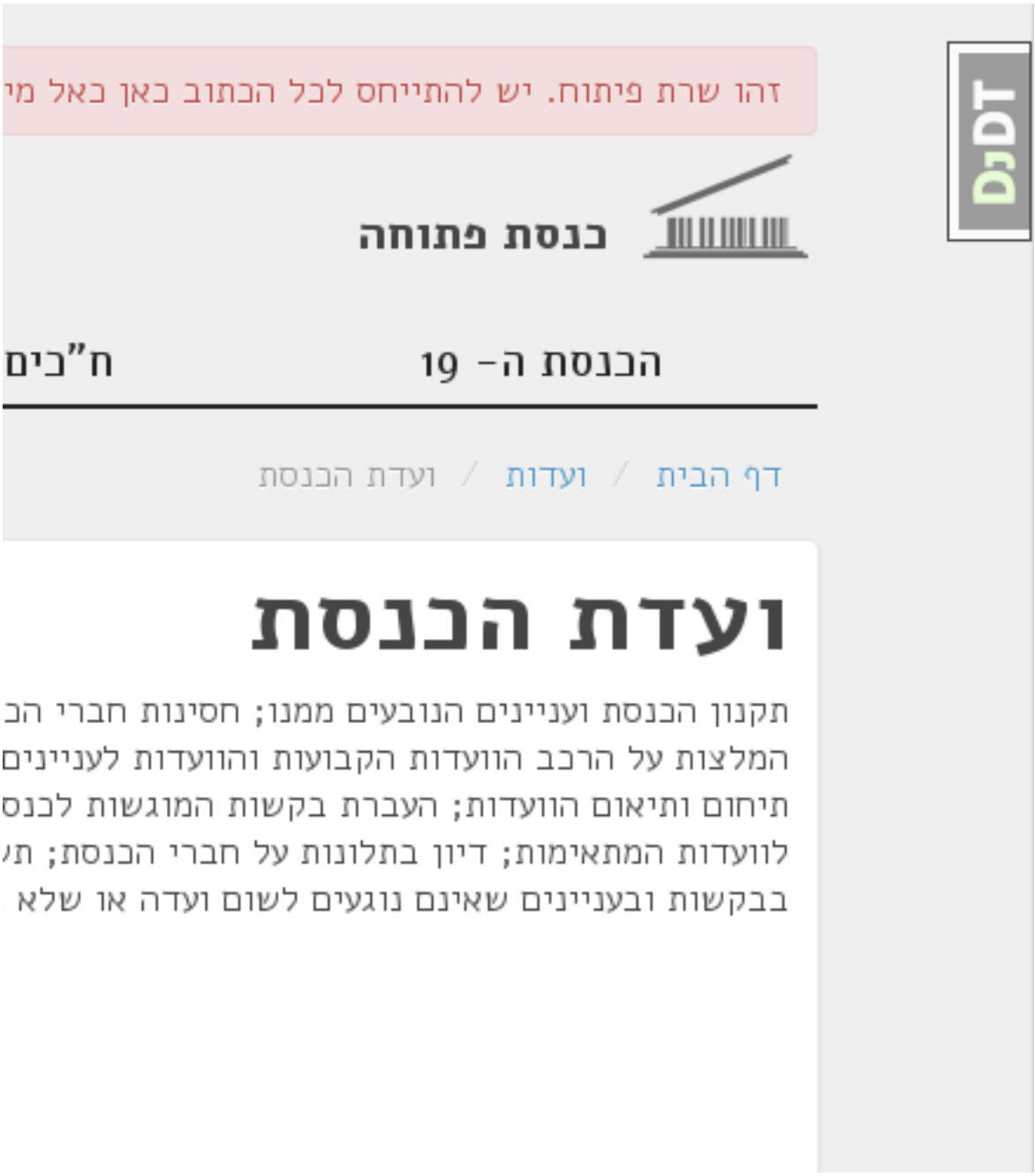
To run the development server:

```
python manage.py runserver
```

Once done, you can access it with your browser via http://localhost:8000 .

Using the debug toolbar

If you've enabled the debug toolbar, you should see it's icon on the top right corner of the page:



Clicking on it will reveal a sidebar which will expose lots of info about the generated page (templates used, context variables, SQL queries etc.).

We're cool ? Time for some *Development Workflow*.

CHAPTER 3

Development Workflow

Congratulations, we have everything installed, now it's time to start working on the project. Here are some guidelines and scenarios to help you get started.

Important:

- MS Windows users: replace `./manage.py` with `python manage.py`
- Run `manage.py` commands from the *Open-Knesset* directory, with the **virtualenv activated**.

Commits and Pull Requests

Make it easier for you and the maintainers, increasing the chances of a pull request getting accepted:

- No big Pull Requests. It makes reviewing and ensuring correctness hard. If possible, break it to smaller commits/pulls, each related to a specific issue.
- Always work on a specific issue from our issue tracker. Open new issue if needed and claim it in the comments.
- Discuss big things in the Open Knesset Developers group.

Before Coding

We need to make sure we're in sync with changes done by others (upstream).

Important: Please do this every time before you start developing:

Update the code and requirements

Enter the *Open-Knesset* directory, and run:

```
git pull git@github.com:hasadna/Open-Knesset.git master
```

Note: Running this command requires having SSH keys registered with github. If you don't have these, or if you don't understand what this means and do not want to look it up, you can use:

git pull https://github.com/hasadna/Open-Knesset.git master

If *requirements.txt* was modified, make sure all of them are installed (no harm running this command even in case of no changes):

pip install -r requirements.txt

Note: We recommend running the pip command from the parent diretory (the virtualenv's root), as it may create an `src` directory when pulling packages from git repos. in that case:

```
`pip install -r Open-Knesset/requirements.txt`
```

Run migrations and tests

```
./manage.py migrate
./manage.py test
# by default the browser tests use firefox - make sure it's installed first
./manage.py test --testrunner=knesset.browser_test_runner.Runner
```

If there are any failures, contact the other developers in the oknesset-dev group to see if that's something you should worry about.

See *Development Tips* for a few bash functions that may help.

While Coding

General

- Write tests for everything that you code.
- Keep performance in mind - test the number of db queries your code performs using `./manage.py runserver` and accessing a page that runs the code you changed. See the output of the dev-server before and after your change.

Adding a field to existing model

We use South to manage database schema migrations. The work process is:

- Add the field you want to model *SampleModel* in app *sample_app*
- *python manage.py schemamigration sample_app –auto* this generates a new migration under sample_app/migrations. You should review it to make sure it does what you've expected.

- *python manage.py migrate* runs the migration against the database.
- Don't forget to **git add** and **commit** the migration file.

After you code

- `./manage.py test` # make sure you didn't break anything
- `./manage.py test --testrunner=knesset.browser_test_runner.Runner` # run the browser tests (see *Browser testing*)
- `git status` # to see what changes you made
- `git diff filename` # to see what changed in a specific file
- `git add filename` # for each file you changed/added.
- `git commit -m "commit message"`

  Please write a sensible commit message, and include "fix#: [number]" of the issue number you're working on (if any).
- `git push` # push changes to git repo
- go to github.com and send a "pull request" so your code will be reviewed and pulled into the main branch, make sure the base repo is **hasadna/Open-Knesset**.

CHAPTER 4

Working on CSS and Documentation

CSS

We're using LESS (no direct editing of CSS). If you'd like to contribute to the design efforts:

Before first run, and only once, you'll need:

```
git submodule init
git submodule update
```

We recommend to use nvm to install the correct node version (so that all the developers get consistent css results):

```
install nvm, see: https://github.com/creationix/nvm
then, run:
$ cd OpenKnesset
OpenKnesset$ nvm install
```

If you encounter problems using nvm, you can install the node another way, you need node in the version specified in .nvmrc file

Install less using the version specified in package.json:

```
$ cd OpenKnesset
OpenKnesset$ nvm use
OpenKnesset$ npm install
```

Make your changes to the files in the `less` directory, and compile (assuming you're in the `Open-Knesset` directory):

```
$ cd OpenKnsset
OpenKnesset$ nvm use
OpenKnesset$ npm run less
```

Documentation

Our documentation is written with Sphinx, install it with the virtualenv activated:

```
pip install sphinx
```

Edit the relevant docs under the `docs` directory, and once done, run `make html`. You'll have the resulting documentation in `build/html` directory.

We have 2 documentation directories:

- api — API and Embedding for 3rd party apps/services developers
- devel — Developer guide for the OpenKnesset project (TBD)

e.g: To work on the devel docs, edit the files under `docs/devel/source`, once ready to build:

```
cd docs/devel
make html
```

You'll have the result under:

```
docs/devel/build/html
```

CHAPTER 5

Development Tips

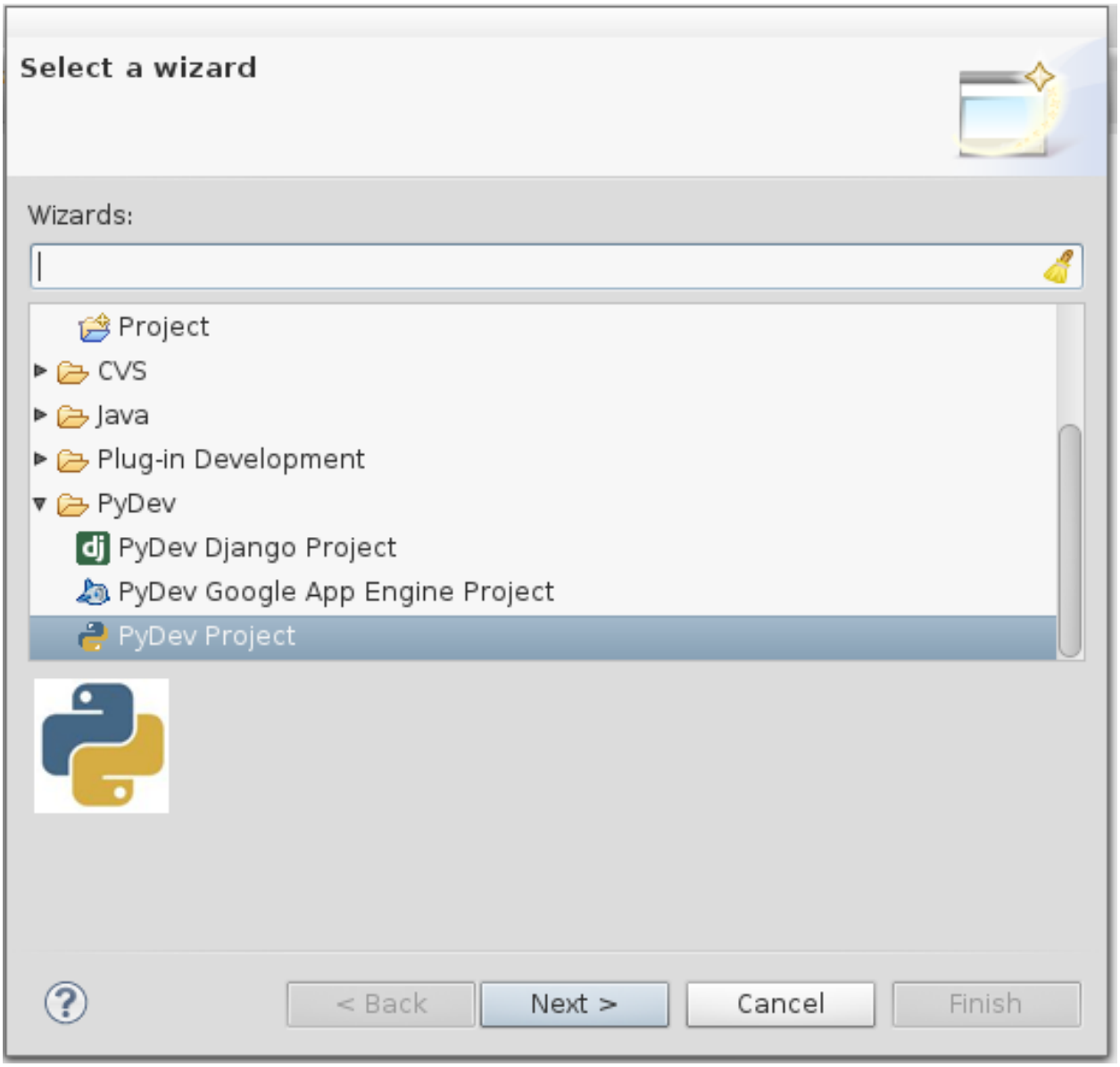
IDE setup

Basically you should lookup for virtualenv configuration of your preferred development environment.

Eclipse with PyDev

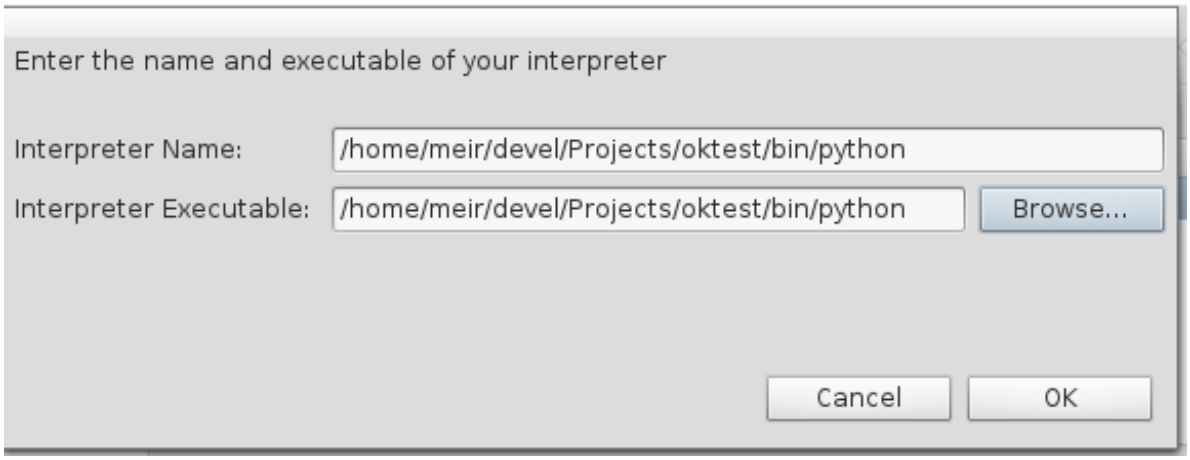
Start by creating a new **PyDev** Project.

> **Warning:** Don't create a Django project! It'll overwrite manage.py and create other files. We'll convert it to a Django project later.



- Click “Next” and type a name for the project — e,g `OpenKnesset`.
- Uncheck “Use default” if checked, click “Browse”, navigate to the virtualenv and selected the previously cloned “Open-Knesset” directory.
- Click the link named “Click here to configure an interpreter not listed”.

  In the opened window, click “New”, and in the opened window “browse”. Navigate to your virtualenv and select the *python* executable in your *Scripts* folder (for MS Windows) or *bin* (on Linux).

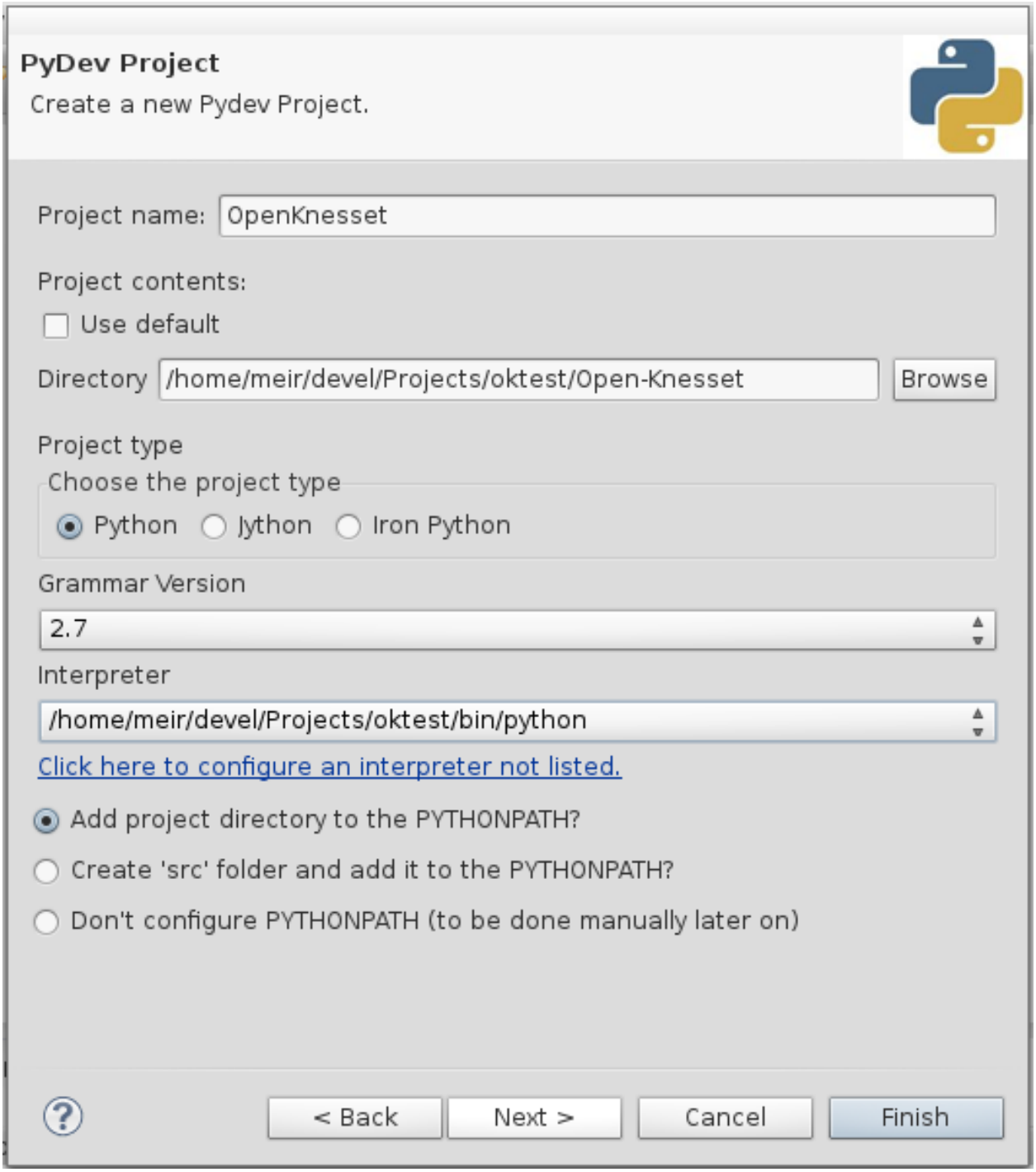


Click “OK”. A window will popup with folders to be added to python path, click “OK”. If you get a warning click “Proceed anyways”. Let it process the libraries and click “OK”.

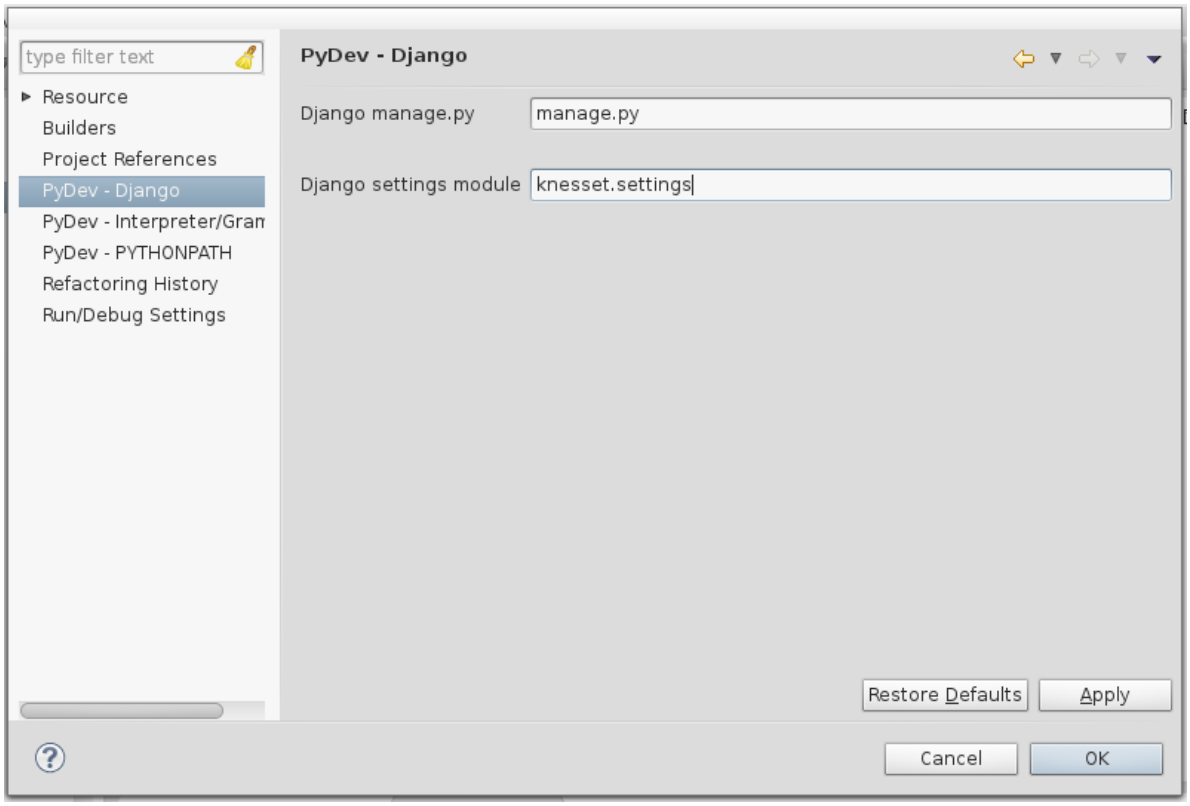
Now you’re back in the Project’s dialog. Select the interpreter you’ve just added.

- Make sure “Add project directory to the PYTHONPATH” is selected.

You should have something like:



- Click “Finish”, and switch to the PyDev perspective (if the dialog appears).
- Now you should have the project in the PyDev Package explorer (left side). In the project. Right click it and select “PyDev” | “Set as Django project”.
- Right click the project and select “Properties”.
- Select “PyDev - Django”, and enter `manage.py` int “Django manage.py” and `knesset.settings` in “Django settings module”.



Click “OK”.

- Create a run configuration for your project (to make sure it’ll find the database, etc.):
  - Right click the project, select “Run As” | “Run configurations”.
  - Right click “PyDev Django” and select “New”.
  - Give it a name (e.g. “oknesset Django”)
  - In the Project field click “Browser” and select you project.
  - In Main Module, click browse and select “manage.py”.
  - In the “Arguments” tab click, in “Program arguments” enter `runserver --noreload`.
  - Click “Run”, you should be able to open your browser and access http://localhost:8000 . You can use this run configration from now on.

Useful bash functions

You can use some variation on these functions when beginning a development session (only after you finished the initial setup). To use these, paste these functions into your `~/.bashrc` file and change the `~/oknesset/` in oknesset_activate to the directory where you installed the project’s virtualencv. These functions will now be available to you in all future bash sessions, but to get it working in already opened tabs, you’ll need to type `source ~/.bashrc`.

A possible session might include:

- Open a bash tab and run `oknesset_update` to pull the new code and db, install any new dependencies and run the tests to check that everything is ok.

- Run `oknesset_runserver` in the same tab to launch the development site.
- Open another tab and run `oknesset_activate` to enter the Open Knesset directory (and virtualenv) to do any manual work on the files.
- Assuming you have the documentation tools set up and have changed some .rst files, run `oknesset_makedocs` to recreate the html pages.

```
oknesset_activate () { #Just enter the Open-Knesset directory and activate the␣
↪virtualenv
    cd ~/oknesset/ #or wherever you put it

    source bin/activate
    cd Open-Knesset
}

oknesset_update () { #Pull the new code and prepare the environment
    oknesset_activate

    DB_URL="http://oknesset-devdb.s3.amazonaws.com/dev.db.bz2"
    if wget --timestamping $DB_URL | grep Saving  # new version downloaded
    then
        mv dev.db dev.db.old
        bzip2 -kd `basename $DB_URL`
    fi

    git pull git@github.com:hasadna/Open-Knesset.git master

    cd ..
    pip install -r Open-Knesset/requirements.txt
    cd Open-Knesset

    ./manage.py migrate
    ./manage.py test
}

oknesset_runserver () { #start the local server
    oknesset_activate
    ./manage.py runserver
}

oknesset_makedocs () { #make the documentation html
    oknesset_activate
    pushd docs/devel >/dev/null
    make html
    popd >/dev/null
}
```

Debugging

During debugging, you can use the Python debugger, pdb.

Add `import pbd` in the code, and `pdb.set_trace()` in the line you want to begin debugging from.

Run the server and browse the site as usual. When the code hit the line you set a trace in, the browser will hang, and you will get the pdb shell back:

```
> /home/yourname/workspace/oknesset/Open-Knesset/agendas/models.py(448)get_mks_
↪values()
-> summaries_for_ranges = []

(Pdb)
```

You can run any python code from there, just like a normal python interpreter, in addition to control commands, like `c` to continue execution, and `n` to step single line.

Warning: The debugger will not work in `python manage.py`, since the output is blocked. The tests will just hang.

Important: Remember to remove any `set_trace()` commands from the code before committing.

Quicker testing

PASSWORD_HASHERS

You may want to add this too to *local_settings.py*, in order to speed up the testing by 25%-30%:

```
import sys
if 'test' in sys.argv:
    PASSWORD_HASHERS = (
        'django.contrib.auth.hashers.MD5PasswordHasher',
    )
```

--failfast

if you want the test suite to fail as soon as the first test does, add `--failfast` flag:

```
python manage.py test --failfast
```

alert

In Ubuntu, you can add an `alert` after this command:

```
python manage.py test --failfast; alert
```

This will pop up a notification when the test is finished.

CHAPTER 6

Browser testing

Running Browser Tests

Browser tests are run with a different test runner, so running ./manage.py test will not run the browser tests.

Instead, you have to specify a different test runner:

```
./manage.py test --testrunner=knesset.browser_test_runner.Runner
```

This test runner looks for files that start with browser_cases and runs only the tests contained in those files.

You can run the tests locally - on your own browser, or remotely:

Running tests locally

By default the tests run locally using Firefox. You can specify a different browser using the –browser parameter.

```
./manage.py test --testrunner=knesset.browser_test_runner.Runner --browser=Chrome
```

The tests are run using selenium, check the selenium docs for available browser options: http://selenium-python.readthedocs.org/api.html

Running tests remotely - using SauceLabs

A better option for running tests is using the SauceLabs service (https://saucelabs.com/). You will first need a username and access key - we have an open source license for SauceLabs but to avoid misuse we can't provide it here, ask your fellow Open Knesset developer for this.

To use SauceLabs you first need to setup a tunnel from your local server to their remote service. This is done using Sauce Connect, see the SauceLabs docs for how to install and use it: https://docs.saucelabs.com/reference/sauce-connect/

Once you have the Sauce Connect tunner working using the username and accesskey, you can run the tests:

```
./manage.py test --testrunner=knesset.browser_test_runner.Runner --browser=Sauce --
↪sauce-user=OpenKnesset --sauce-accesskey=ACCESS_KEY
```

In the test output you can find links to the test sessions, something like this: https://saucelabs.com/jobs/0383bb8bb3094a35b36edfd4e68aa094

Note that all the test sessions are publicly available because we have an open source license.

Writing tests

Tests are written using the python selenium bindings and the standard django unit tests framework.

All browser test file names must be in the format browser_cases*.py

You can see a simple test in knesset/browser_cases.py - it is commented with useful details.

Refer to the selenium python documentation for help on using selenium: http://selenium-python.readthedocs.org

CHAPTER 7

Scraping

One of the most important tasks in Open Knesset are the scraping processes, this document will try to describe all those processes (but pay attention that as the code changes, this documentation might not..)

We are currently moving towards having knesset-data-django project handle all the scraping jobs.

See the project documentation for more details: https://github.com/hasadna/knesset-data-django/blob/master/README.md

Knesset-data-django is integrated in Open Knesset, so you can run the management commands detailed there in Open Knesset.

Scraping Tasks

Votes

Votes are scraped using the syncdata process, specifically the "update_votes" function.

This functions interates based on vote id corresponding to knesset's vote page, e.g. http://www.knesset.gov.il/vote/heb/Vote_Res_Map.asp?vote_id_t=23556

For the above URL, the vote id is 23556.

We then scrape all the relevant data from that html page, and connect to related objects.

Committees

Committees are scraped using the Knesset-data-django project.

See their documentation for details: https://github.com/hasadna/knesset-data-django/blob/master/README.md

Presence

presence data is acumulated using the presence/PresenceChecker.sh script which is run by cron every 6 hours

this process goes to http://www.knesset.gov.il/presence/eng/PresentList_eng.aspx and checks which mks are currently in knesset

Where and how the tasks are run

The scraping tasks are all run from the db server using cron, these are the cron jobs (last updated: ~2015):

```
1 02,14 * * * /oknesset_data/oknesset/Open-Knesset/manage.py update_all_feeds
05 06,12,18 * * * /oknesset_data/presence/PresenceChecker.sh
30 03 * * * /oknesset_data/oknesset/Open-Knesset/manage.py update_videos --only-
↪members --current-knesset
45 03 * * * /oknesset_data/oknesset/Open-Knesset/manage.py parse_plenum_protocols --
↪download --parse
00 04 * * * /oknesset_data/oknesset/Open-Knesset/manage.py parse_future_plenum_
↪meetings
15 04 * * * /oknesset_data/oknesset/Open-Knesset/manage.py syncdata --update
59 04 * * * /oknesset_data/oknesset/Open-Knesset/manage.py send_email_to_editors
00 05 * * * /oknesset_data/oknesset/Open-Knesset/manage.py notify --daily
01 05 * * 5 /oknesset_data/oknesset/Open-Knesset/manage.py notify --weekly
02 01,05,09,13,17,21 * * * /oknesset_data/oknesset/Open-Knesset/manage.py send_mail
03 05 * * * /oknesset_data/oknesset/Open-Knesset/manage.py parse_future_committee_
↪meetings
30 04 * * * /oknesset_data/oknesset/Open-Knesset/manage.py okscrape lobbyists --dblog
```

CHAPTER 8

DevOps - Servers, Configuration, Deployment, Common tasks

Servers

Following is the servers configuration (as it was on Feb. 16, 2016):

db1 DB server, runs cronjobs, (EC2 Name: Postgres1)

small1 web app

small2 web app

Configuration

web app servers Runs supervisor service which includes oknesset app

The file /etc/supervisor/conf.d/oknesset.conf contains the relevant configuration

This is the command that it runs: command=newrelic-admin run-program gunicorn knesset.wsgi:application -w 4 -t 60

db servers Runs the DB Runs cronjobs, see https://github.com/hasadna/Open-Knesset/blob/master/deploy/crontab.txt

Deployment

Deployment is done using fabric, see: https://github.com/hasadna/Open-Knesset/blob/master/fabfile.py

There is a local_fab_settings.py file which contains login details.

common deployment tasks

$ fab deploy_backend deploy to the db server

$ fab deploy_backend:migration=yes deploy to db and run ./manage.py migrate as well

$ fab deploy_backend:requirements=yes,migration=yes deploy to db, run migrations and also pip install -r requirements.txt

$ fab deploy_web deploy to the web servers (small1, small2)

$ fab deploy_web:requirements=yes deploy to the web servers and also run pip install -r requirements.txt

Common Tasks

Updating the dev DB

Run the following on the production DB instance:

- (oknesset) Open-Knesset$ ./manage.py sync_dev
- (oknesset) Open-Knesset$ bzip2 dev.db -fk
- (oknesset) Open-Knesset$ s3put –access_key AWS_ACCESS –secret_key AWS_SECRET –bucket oknesset-devdb dev.db.bz2
- (oknesset) Open-Knesset$ zip dev.db.zip dev.db
- (oknesset) Open-Knesset$ s3put –access_key AWS_ACCESS –secret_key AWS_SECRET –bucket oknesset-devdb dev.db.zip
- (oknesset) Open-Knesset$ rm dev.db dev.db.bz2 dev.db.zip

After the above is done, you may need to log-in to S3 and make the files public (right-click each file and choose “make public”)

CHAPTER 9

Feature toggles

Sometimes we would like Develop new features but not expose them yet to users. Perhaps we need to expose them only to Qa, or by user role (such as admins).

Maybe we need some backend model to start gathering data and calculations from real production data but not expose this data in the view, while retaining the possibilty

Another use case can be rolling and exposing a feature only to a small sample of our users. We would like to get input from real usage pattens before we proceed with rolling a feature to all users..

For all those use cases we can use feature flags/feature toggle.

A feature toggle can toggled on or off, globally or sometimes according to a specific condition.

You can (and should) read more about the feature toggle pattern in Martin Fowler's blog

Using "waffle" - a feature toggle framework for django

We are currently using Django-Waffle as a feature toggle framework. It allows using switches and flags (switches that accept the request and allow configuring the on/off flag according to the user, request, environ and more) You can use flags, switches and other *waffle* goodies in templates, views, javascript etc. You can read more about operating waffle in Waffle docs

Note that when you add a feature toggle in the code without adding in the db (through admin or command) then
The feature toggle is considered closed

Cleaning up after

A common problem with feature toggles is forgetting to remove them when finished. after some time passes nobody remembers anymore the original purpose of the toggle and no ones dare removing it. Even worst things can happen Please remove feature toggles when not needed any more.

CHAPTER 10

Project Management

Pull Request Code Review

All code contributions are sent using a GitHub Pull Request, for example: https://github.com/hasadna/Open-Knesset/pull/556

When you start reviewing a pull request, assign it to yourself so that we won't have 2 people reviewing the same code together.

You should review both the code, but also the change itself, make sure it fits Open Knesset and our needs.

If in doubt, raise the issue on the forum or in slack.

Keep in mind that when dealing with volunteers, you should make sure they feel welcome to contribute more code.

Merging an approved pull request

After a pull request is approved, you can merge it. Currently, this is done right after merging by the same person.

So, to merge, you just click the "Merge" button in the pull request (assuming you have write permissions).

Once it's merged to master, you should also update the release notes: https://github.com/hasadna/Open-Knesset/releases

If there is an existing "Draft" release - you can add it to that release.

If there is no "draft" release" - you can create a new release, just bump the relevant number in the version.

Look at a few previous releases to see what's written in the release notes, and how the versions are numbered.

Generally, release notes are meant for non technical people as well, so you should write a few words about the changes that were introduced. Also, you can write deployment notes if any special actions should be performed before or after the deployment.

Deploying a release

Once you have some pull requests merged you might want to deploy the release. The release should be published before deployment (click "Publish" button in GitHub edit release page). This allows us to know which version was deployed. For technical details about how to deploy see the DevOps documentation.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search