
Junction

Junction Developers

Aug 06, 2020

CONTENTS:

1	Development	1
1.1	Getting Started	1
2	Junction (old)	5
2.1	API	5
2.2	Release Notes	9
2.3	Add conference moderators	11
2.4	Setup	11
2.5	Contributing	12
2.6	License	12
3	Indices and tables	13

DEVELOPMENT

Junction is a volunteer maintained open source project and we welcome contributions of all forms. The sections below will help you get started with development, testing, and documentation.

1.1 Getting Started

We're pleased that you are interested in working on Junction.

This document is meant to get you setup to work on Junction and to act as a guide and reference to the the development setup. If you face any issues during this process, please [open an issue](#) about it on the issue tracker.

1.1.1 Initial Setup

Junction's development workflow is automated using `nox`. Thus, you need the `nox` command to be installed on your system. We recommend using `pipx` to install `nox` in its own isolated environment.

```
$ python -m pip install pipx
$ pipx install nox
```

You will need to have a working Redis server on your system. You may additionally want to install Postgres, although it is optional.

Note: On Debian based systems, these can be installed using:

```
$ sudo apt install redis-server postgres
```

Backend

Create a "settings" file for local development with Django.

```
$ cp settings/dev.py.sample settings/dev.py
```

Create the database structure and populate it with sample data.

```
$ nox -- migrate --noinput
$ nox -- sample_data
```

Admin Access

When sample data is generated with `nox -- sample_data`, a superuser is created with the username `admin` and password `123123`.

Frontend

Working on Junction's frontend requires [NodeJS](#) and [yarn](#) to be installed on your system. The frontend is built using [grunt](#). To setup the working environment, run the following:

```
$ cd junction/static
$ yarn install
```

1.1.2 Development workflow

Frontend Autobuilding

Junction has a Grunt configuration that is useful when working on the frontend. The following command starts a build watcher which rebuilds the frontend on every file change.

```
$ grunt
```

For ease of development `app.css` is checked in to the source code. It is not recommended to directly make changes to `app.css`, rather update the less files and run `grunt`, then commit `app.css`

Invoking `manage.py`

Junction's `nox` configuration is set up to invoke `manage.py` when no other session (i.e. `-s ...`) is specified. This also automatically sets up an isolated environment that contains the dependencies of Junction.

```
$ nox # equivalent to 'python manage.py'
$ nox -- runserver # equivalent to 'python manage.py runserver'
$ nox -- migrate # equivalent to 'python manage.py migrate'
```

Running tests

For running the tests, run:

```
$ nox -s test
```

Running linters

For running the linters, run:

```
$ nox -s lint
```

Building documentation

For building the documentation, run:

```
$ nox -s docs
```


JUNCTION (OLD)

NOTE: This document and linked sections may be out of date.

2.1 API

NOTE: This document and linked sections may be out of date.

Junction provides API to access information about the conference, schedule, and feedback. The API is for mobile clients to assist conference attendees. All the request and response format is application/json.

- Demo site: <http://junctiondemo.herokuapp.com/>

2.1.1 Conference - List

- Endpoint: `/api/v1/conferences/`
- Allowed Method: GET.
- Returns all conferences. There is no pagination.
- Sample Response:

```
[ { "id": 5, "name": "PyCon India 2016", "slug": "2016", "description": "...", "start_
↪date": "2016-09-23", "end_date": "2016-09-25", "status": "Proposal submission closed
↪", "venue": "http://in.pycon.org/cfp/api/v1/venues/2/" }, ]
```

- venue key holds URL of the venue details.

2.1.2 Venue - List

- Endpoint: `/api/v1/venues/`
- Allowed Methods: GET
- List all the venues.
- Sample Response:

```
[ { "id": 1, "name": "Nihmans", "address": " Hosur Road, Lakkasandra, Behind Bus Stop,
↪ Bengaluru, Karnataka 560030", "latitude": "12.943112200000000", "longitudes": "77.
↪59686430000000000" },  ]
```

(continues on next page)

2.1.3 Venue - Detail

- Endpoint: `/api/v1/venues/<id>/`
- Allowed Method: GET
- Return specific venue details.
- Sample Response:

```
{ "id": 1, "name": "Nihmans", "address": " Hosur Road, Lakkasandra, Behind Bus Stop, ↵  
↵Bengaluru, Karnataka 560030", "latitude": "12.943112200000000", "longitudes": "77.  
↵5968643000000000" }
```

2.1.4 Room - List

- Endpoint: `/api/v1/rooms/`
- Allowed Method: GET
- List all rooms of all venues..
- Sample Response:

```
[ { "id": 4, "name": "Buffet Area", "venue": "http://in.pycon.org/cfp/api/v1/venues/1/  
↵", "note": "Left end of the ground floor" },..]
```

2.1.5 Room - Venue specific

- Endpoint: `/api/v1/rooms/?venue=<id>`
- Display list of rooms in the venue.
- Allowed Method: GET
- Sample Response:

```
[ { "id": 4, "name": "Buffet Area", "venue": "http://in.pycon.org/cfp/api/v1/venues/1/  
↵", "note": "Left end of the ground floor" }, { "id": 3, "name": "Room 3", "venue":  
↵"http://in.pycon.org/cfp/api/v1/venues/1/", "note": "Ground Floor" }, { "id": 2,  
↵"name": "Room 2", "venue": "http://in.pycon.org/cfp/api/v1/venues/1/", "note":  
↵"Ground Floor" }, { "id": 1, "name": "Room 1", "venue": "http://in.pycon.org/cfp/  
↵api/v1/venues/1/", "note": "Ground Floor" } ]
```

2.1.6 Schedule: List

- Endpoint: /api/v1/schedules/
- Allowed Methods: GET
- All the schedule items of all conferences.
- Sample Response:

```
{ "2015-10-04": { "08:30:00 - 09:15:00": [ { "conference": "http://in.pycon.org/cfp/
↪api/v1/conferences/1/", "session": { "description": "" }, "room_id": 4, "end_time":
↪"09:15:00", "event_date": "2015-10-04", "start_time": "08:30:00", "type": "Break",
↪"id": 37, "name": "Registration & Breakfast" } ], "09:30:00 - 10:15:00": [ {
↪"conference": "http://in.pycon.org/cfp/api/v1/conferences/1/", "session": {
↪"description": "" }, "room_id": 1, "end_time": "10:15:00", "event_date": "2015-10-04
↪", "start_time": "09:30:00", "type": "Talk", "id": 38, "name": "Keynote - Nicholas
↪H.Tollervey" } ] },
```

- Response data keys are conference day date and its schedule grouped by time.
- 2015-10-04 - Conference day date.
- 08:30:00 - 09:15:00 - Start of the session - End of the session.
- session dictionary contains details about the session.
- type: Type of the session like break, lunch, talk, workshop etc...
- room_id: Place of the session. The Client should use API or cached data to fetch room name.

2.1.7 Schedule: List of sessions for the conference

- Endpoint: /api/v1/schedules/?conference=<conference_id>
- Allowed Methods: GET
- All the schedule items of the conference.
- Sample Response:

```
{ "2015-10-04": { "08:30:00 - 09:15:00": [ { "conference": "http://in.pycon.org/cfp/
↪api/v1/conferences/1/", "session": { "description": "" }, "room_id": 4, "end_time":
↪"09:15:00", "event_date": "2015-10-04", "start_time": "08:30:00", "type": "Break",
↪"id": 37, "name": "Registration & Breakfast" } ], "09:30:00 - 10:15:00": [ {
↪"conference": "http://in.pycon.org/cfp/api/v1/conferences/1/", "session": {
↪"description": "" }, "room_id": 1, "end_time": "10:15:00", "event_date": "2015-10-04
↪", "start_time": "09:30:00", "type": "Talk", "id": 38, "name": "Keynote - Nicholas
↪H.Tollervey" } ] }, }
```

2.1.8 Device - Register

- Endpoint: /api/v1/devices/
- Allowed Method: POST
- Junction accepts feedback via API. We support anonymous feedback, but the device registration is mandatory. Registered device can only submit the feedback.
- Payload: {'uuid': 'uuid-1'}.
- Response: If UUID exists status code is 400 else the status code is 201 with data.

2.1.9 Feedback Questions - List

- Endpoint: /api/v1/feedback_questions/?conference_id=<id>
- Allowed Method: GET
- Fetch all feedback questions for the conference.
- Sample Response:

```
{ "Workshop": { "text": [ { "schedule_item_type": "Workshop", "is_required": false,
↪ "type": "text", "id": 2, "title": "Any other feedback for workshop ?" } ], "choice
↪ ": [ { "title": "Does the speaker have experience on the subject?", "schedule_item_
↪ type": "Workshop", "allowed_choices": [ { "id": 15, "value": 2, "title": "Good" },
↪ { "id": 14, "value": 1, "title": "Ok" }, { "id": 13, "value": 0, "title": "Bad" } ],
↪ "is_required": true, "type": "choice", "id": 5 }, { "title": "How hands on was the
↪ workshop ?", "schedule_item_type": "Workshop", "allowed_choices": [ { "id": 6,
↪ "value": 2, "title": "Good" }, { "id": 5, "value": 1, "title": "Ok" }, { "id": 4,
↪ "value": 0, "title": "Bad" } ], "is_required": true, "type": "choice", "id": 2 }, {
↪ "title": "How was the content ?", "schedule_item_type": "Workshop", "allowed_choices
↪ ": [ { "id": 3, "value": 2, "title": "Good" }, { "id": 2, "value": 1, "title": "Ok"
↪ }, { "id": 1, "value": 0, "title": "Bad" } ], "is_required": true, "type": "choice",
↪ "id": 1 } ] }, "Talk": { "text": [ { "schedule_item_type": "Talk", "is_required":
↪ false, "type": "text", "id": 1, "title": "Any other feedback for the talk ?" } ],
↪ "choice": [ { "title": "Does the speaker have experience on the subject?",
↪ "schedule_item_type": "Talk", "allowed_choices": [ { "id": 18, "value": 2, "title":
↪ "Good" }, { "id": 17, "value": 1, "title": "Ok" }, { "id": 16, "value": 0, "title":
↪ "Bad" } ], "is_required": true, "type": "choice", "id": 6 }, { "title": "How was
↪ the presentation ?", "schedule_item_type": "Talk", "allowed_choices": [ { "id": 12,
↪ "value": 2, "title": "Good" }, { "id": 11, "value": 1, "title": "Ok" }, { "id": 10,
↪ "value": 0, "title": "Bad" } ], "is_required": true, "type": "choice", "id": 4 }, {
↪ "title": "How was the content ?", "schedule_item_type": "Talk", "allowed_choices":
↪ [ { "id": 9, "value": 2, "title": "Good" }, { "id": 8, "value": 1, "title": "Ok" },
↪ { "id": 7, "value": 0, "title": "Bad" } ], "is_required": true, "type": "choice",
↪ "id": 3 } ] } }
```

- Response data keys are session types. Workshop and Talk session type questions are values of the respective key. Each of the session types contains two keys, text and choice. text dictionary contains text type questions and choice dictionary contains choice based questions. Both the question type has some common fields. id is the unique identifier of the question. title is the displayable text of the question. is_required is the boolean field, true means feedback should have an answer for the item. choice questions have allowed_choices, which contains allowed values. Each item is a dictionary containing id, title, value. Use title to display the answer.

2.1.10 Feedback submission

- Endpoint: `/api/v1/feedback`
- Allowed Methods: POST.
- Header: 'Token: <registered_device_uuid>'.
- Sample Payload:

```
{'text': [{'text': 'Ok', 'id': 1}], 'schedule_item_id': 1, 'choices': [{'id': 1,
↪ 'value_id': 1}]}
```

- `schedule_item_id` is the id of the session accepting feedback. The payload contains text and choices answers.
- When request succeeds, the status code is 200 and when the request fails, the status code is 400 or 403. When the input data is incorrect code is 400 and 403 when device token is missing.
- Sample success response: 201

```
{'text': [{'text': 'Ok', 'id': 1}], 'schedule_item_id': 1, 'choices': [{'id': 1,
↪ 'value_id': 1}]}
```

- Sample failure response: 400

```
{'choices': [{u'non_field_errors': [u"The multiple choice value isn't associated with_
↪ question"]}]}
```

- Sample failure response: 403

```
{u'detail': u'Authentication credentials were not provided.'}
```

2.2 Release Notes

NOTE: This document and linked sections may be out of date.

2.2.1 0.4.0

- Fix filters on the proposal review form #418 (@ChillarAnand)
- Update proposal urls to use hashids. The slug will now update itself when the title of proposal changes.
- Conference duration is displayed in a short form.
- Add ability to remove a casted vote on a proposal.
- ...

2.2.2 0.3.0

Date: 31st March 2016

Features

- conference moderators can be added per conference using a `conference_moderator` management command.
- Improved conference admin.
- Basic REST API for accessing conference details and proposals.
- REST API to submit feedback for a talk/workshop.
- UI updates

Fixes

- fix the tag filtering for all the proposal sections inside proposal list (#170)
- Update proposal detail styling (#207)

2.2.3 0.2.0

Date: 29th March 2015

Added

- add support for fig based development environment (#129)
- django admin got a new theme based on django-flat-theme
- setup social sharing on proposal detail pages (#185)
- add `SITE_URL` settings to support path based root url of site.
- add docker/fig setup
- add celery support
- send mail to reviewers for new proposals

Changes

- hide reviewer name in comments (#193)
- UI: remove 'description' from page proposals list (#149)
- update styling of create proposal forms

Fixes

- upgrade django and other libraries to latest
- fix incorrect login url on comment section

2.2.4 0.1.0

Date: 8th February 2015

- initial release with core functionality working

2.3 Add conference moderators

NOTE: This document and linked sections may be out of date.

First add users to `Proposal Reviewers` model and assign them to `Proposal Sections` as reviewers.

- Assign unique name to reviewer. This name will be shown in review comments to proposers. All reviewers can see other reviewers comments. Section reviewers can only add reviews.

2.3.1 Reviewer Comments

On the proposal page, reviewers can see all `Reviews` and `Reviewers Talk` section.

Version: 0.2.0-dev

Junction is a software to manage proposals, reviews, schedule, feedback during conference.

2.4 Setup

It is advised to install all the requirements inside `virtualenv`, use `virtualenvwrapper` to manage virtualenvs.

```
pip install -r requirements.txt
cp settings/dev.py.sample settings/dev.py
python manage.py migrate --noinput
python manage.py sample_data
```

Initial auth data: admin/123123

2.4.1 Configuring Django-allauth

- Go to `http://localhost:8000/nimda/sites/site/`
- Change the default site's(the one with ID = 1) name and display to `localhost:8000`
- Go to `Social Applications` in admin panel and add `Github` and `Google's` auth details

2.4.2 Making Frontend Changes

Make sure you have `nodejs`, `npm`, `bower`, `grunt-cli` & `grunt` installed

```
$ cd junction/static
$ npm install
$ bower install
$ grunt // This starts a watcher to watch for file changes
```

2.5 Contributing

1. Choose an [issue](#) and ask any doubts in the issue thread.
2. Report any bugs/feature request as [github new issue](#), if it's already not present.
3. If you are starting to work on an issue, please leave a comment saying "I am working on it".
4. Once you are done with feature/bug fix, send a pull request according to the [guidelines](#).

2.6 License

The MIT License (MIT)

Copyright (c) 2014-15 Python Software Society of India

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`