# BYCEPS

Jochen Kupperschmidt

Aug 01, 2022

# CONTENTS

**BYCEPS** is the *Bring-Your-Computer Event Processing System*.

It is a tool to prepare and operate a LAN party, both online on the Internet and locally as an intranet system, for both organizers and attendees.

The system incorporates both experience from more than 15 years of organizing LAN parties as well as concepts and source code developed for more than a decade.

CHAPTER

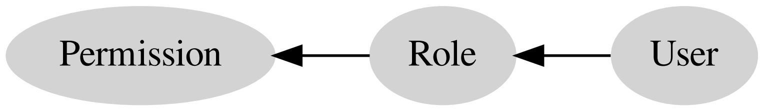
# ONE

# CONCEPTS

## 1.1 Authorization

User authorization in BYCEPS is based upon permissions and roles.

### 1.1.1 Structure



- A **permission** is a requirement to perform a specific action.

  A permission can be part of one or more roles.
- A **role** is a set of permissions that can be granted to users.

  Permissions can only be assigned to roles, but not directly to users. Roles are the links between permissions and users.

  Multiple roles may contain the same permission, but preferably shouldn't (see *example*).
- A **user** can be assigned one or more roles through which the associated permissions are granted.

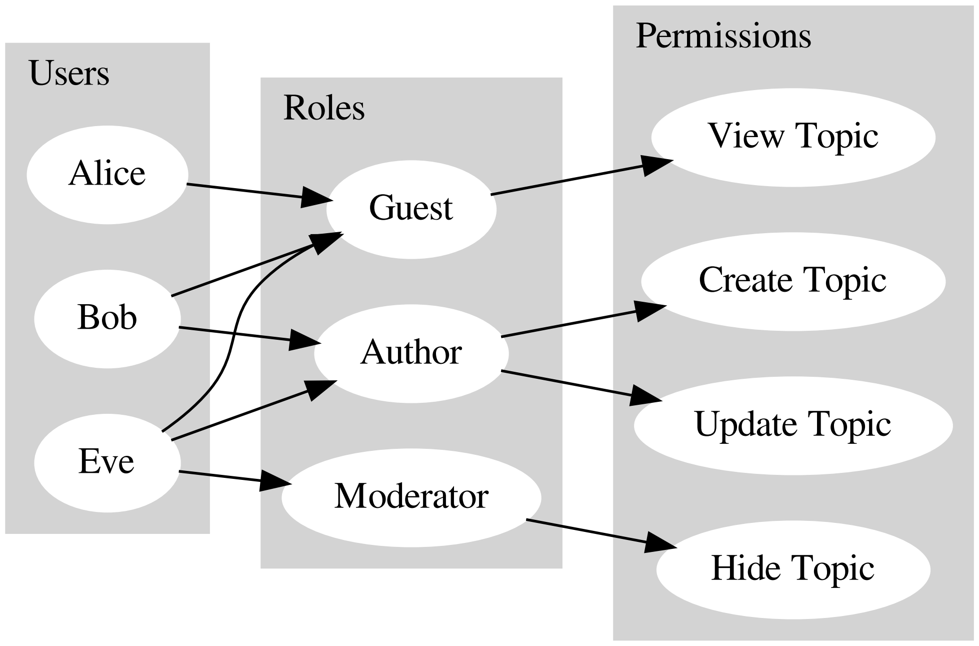
### 1.1.2 Rationale

This design was chosen because

1. it simplifies authorizing users to do predefined sets of actions by leveraging and combining existing roles
2. while making it easy to customize the permissions a user should have by simply adding specific roles with very few permissions instead of having to copy and slightly adjust entire roles with dozens of permissions (as it would be required in a system that only allows to assign a single role to a user).

### 1.1.3 Example

This example demonstrates how board-related permissions can be grouped into roles. Those roles are then combined per user to provide the permissions that should be granted.



As a result, the users have these roles and permissions:

| User | Roles | Permissions |
|---|---|---|
| Alice | • Guest | • View Topic |
| Bob | • Guest<br>• Author | • View Topic<br>• Create Topic<br>• Update Topic |
| Eve | • Guest<br>• Author<br>• Moderator | • View Topic<br>• Create Topic<br>• Update Topic<br>• Hide Topic |

## 1.2 Blueprints

BYCEPS is structured using Flask's Blueprints.

A blueprint acts as a namespace and container for functionality of a certain topic.

It bundles:

- server-side code (Python, `*.py`),
- templates (Jinja, `*.html`),
- and static files, including:
  - front-end styles (CSS, `*.css`)
  - front-end behaviour (JavaScript, `*.js`)
  - images (`*.jpeg`, `*.png`, etc.)

Blueprints should use their own database tables instead of extending or modifying existing ones.

Generally, blueprints should be self-contained. This should make it easy to add them to an application, and to disable unwanted ones.

In order to add functionality to BYCEPS, developers are encouraged to wrap their extensions in a blueprint. This makes it easier to keep the base system updated without having to worry about conflicts with their additions. It also makes it easier to distribute their extensions to other interested BYCEPS users.

### 1.2.1 Integration

To fulfill their purpose, blueprints will need to be integrated into the system one way or another.

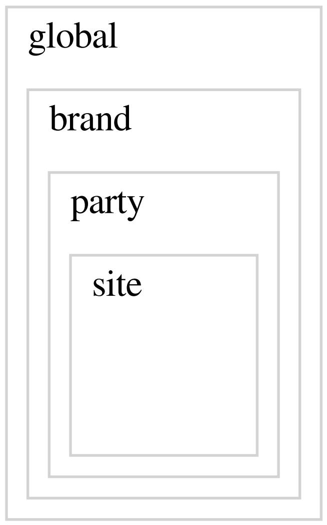
A blueprint may build entirely upon the existing system, and just require a few URL references to be inserted in the navigation or some templates of the base system.

If a blueprint should react on certain events, it can connect to the available *signals*.

## 1.3 Scopes

BYCEPS distinguishes four scopes:

- *global*
- *brand*
- *party*
- *site*



Each entity belongs to exactly one of these scopes.

### 1.3.1 Global

The global scope is the outermost one.

Entities that belong to the global scope include users, roles, permissions, user badges, *brands*, and optionally snippets.

### 1.3.2 Brand

A brand is the identity of a series of parties.

Each brand is part of the *global* scope.

Entities that belong to the brand scope include orga flags, terms of service versions, news channels, boards, *parties*, and optionally snippets.

### 1.3.3 Party

The party scope is for entities that belong to a single party (and are not better situated in the *site scope*).

Each party belongs to a *brand*.

Entities that belong to the party scope include orga teams, shops, tickets, seating areas, and *sites*.

### 1.3.4 Site

The site scope is the innermost one.

Each site belongs to a *party*.

Entities that *can* belong to the site scope include snippets.

## 1.4 Signals

BYCEPS makes use of signals (based on the Blinker package) to provide hooks for specific events.

For example, a signal is emitted every time

- a user account is created
- a topic in the board is created
- an order is placed in the shop

Besides representing the information *that* something happened, signals can (and usually do) contain relevant objects as well.

To receive signals, handlers can be registered for those they are interested in.

Some specific knowledge is necessary to attach code to a specific signal and access its payload, though.

- to import it: the module and name of the signal
- to handle it: the types of the objects it contains, and the keyword argument names they can be accessed with

### 1.4.1 Example

As a simple example for learning purposes, here is the code to print a message to STDOUT (visible when manually starting the application from the command line, e.g. for development and debugging).

```
from byceps.events.board import BoardTopicCreated
from byceps.signals.board import topic_created

@topic_created.connect
def celebrate_created_topic(sender, *, event: BoardTopicCreated = None) -> None:
    print(f'A topic titled has been created: {event.url}')
```

More useful reactions include:

- announcing selected events via email, on IRC, or on social media sites
- creating/assigning a party ticket once the corresponding order has been paid
- running spam detection on new board topics and postings

CHAPTER

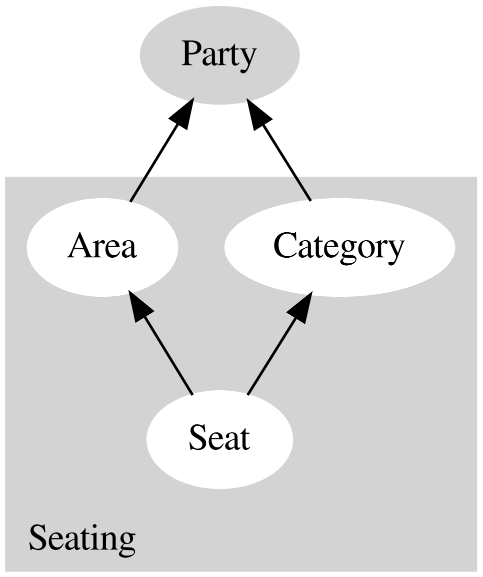
TWO

# AVAILABLE BLUEPRINTS

## 2.1 Seating

BYCEPS' seating model was designed to be flexible enough to fit both *small parties* (say, less than a hundred seats in a single hall) as well as *big ones* (like Northern LAN Convention – NorthCon – with around 3.500 seats).

### 2.1.1 Structure



Each seat references these two entities:

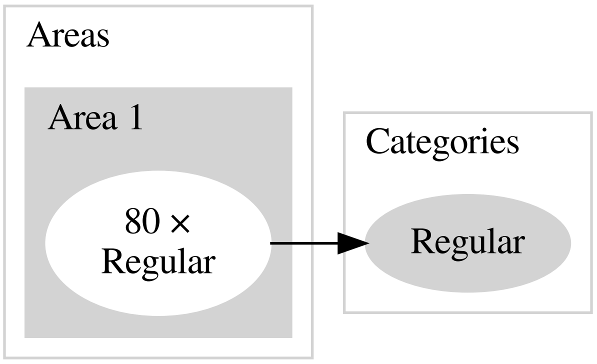
- An **area** represents the physical location of a group of seats.
- A **category** is meant to separate seats in different price ranges from each another.

  Since a ticket is bound to a category, a user with a ticket from category X cannot reserve a seat that belongs to category Y.

Each area and category belongs to a specific party since each seating setup often is party-specific (even if multiple parties are held in the same location).

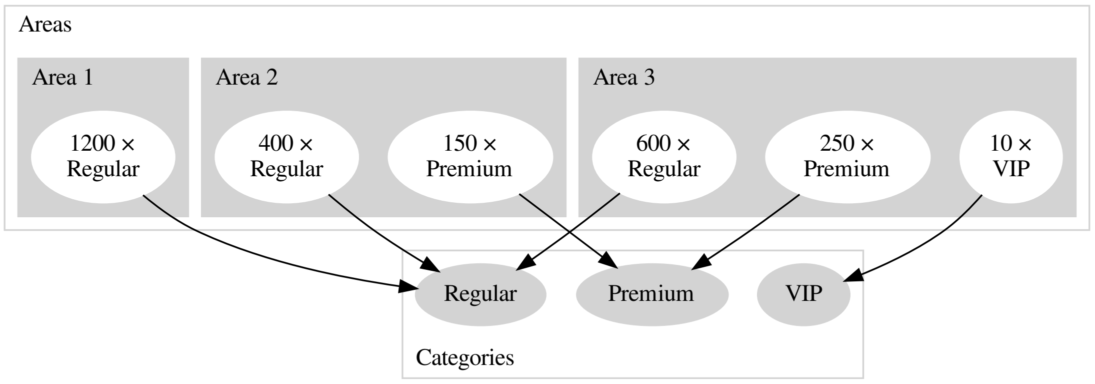
### 2.1.2 Example: Small Party

A small party may take place in a single room or hall, and no distinction is made between the seats in it. Thus, a single area as well as a single category are sufficient, so every seat belongs to the same area and the same category.



### 2.1.3 Example: Big Party

This is a setup for a party that is held in multiple halls and which offers seats in multiple price (and feature) ranges.



# CHAPTER THREE

# INSTALLATION

## 3.1 Requirements

- Python 3.7 or higher
- PostgreSQL 11 or higher
- Redis 5.0 or higher
- Git (for downloading and updating BYCEPS, but not strictly for running it)

## 3.2 Install Debian Packages

Debian Linux is the recommended operating system to run BYCEPS on.

To install packages, become the `root` user (or prefix the following commands with `sudo` to obtain superuser permissions):

```
$ su -
```

Update the list of packages before installing any:

```
# aptitude update
```

On Debian "Bullseye" 11 or Debian "Buster" 10, install these packages:

```
# aptitude install git postgresql python3 python3-dev python3-venv redis-server
```

Additional required packages should be suggested for installation by the package manager.

Refer to the Debian documentation for further details.

## 3.3 Obtain BYCEPS

Grab a copy of BYCEPS itself. For now, the best way probably is to clone the Git repository from GitHub:

```
$ git clone https://github.com/byceps/byceps.git
```

A new directory, `byceps`, should have been created.

This way, it should be easy to pull in future updates to BYCEPS using Git. (And there currently are no release tarballs anyway.)

## 3.4 Set Up a Virtual Python Environment

The installation should happen in an isolated Python environment just for BYCEPS so that its requirements don't clash with different versions of the same libraries somewhere else in the system.

Python already comes with the necessary tools, namely virtualenv and pip.

Change into the BYCEPS path and create a virtual environment (named "venv") there:

```
$ cd byceps
$ python3 -m venv venv
```

Activate it (but don't change into its path):

```
$ . ./venv/bin/activate
```

Note that the first dot is the dot command, followed by a relative file name (which is written as explicitly relative to the current path, `./`).

Whenever you want to activate the virtual environment, make sure to do that either in the path in which you have created it using the above command, or adjust the path to reference it relatively (e.g. `../../venv/bin/activate`) or absolutely (e.g. `/var/www/byceps/venv/bin/activate`).

Make sure the correct version of Python is used:

```
(venv)$ python -V
Python 3.9.2
```

It's probably a good idea to update pip to the current version:

```
(venv)$ pip install --upgrade pip
```

Install the Python depdendencies via pip:

```
(venv)$ pip install -r requirements.txt
```

Install BYCEPS in editable mode to make the `byceps` command as well as the package of the same name available:

```
(venv)$ pip install -e .
```

## 3.5 Create BYCEPS Configuration File

To run BYCEPS, a configuration file is required. Those usually reside in `/config`.

There are two examples, `development-example.py` and `production-example.py`, that you can use as a base for your specific configuration.

For starters, create a copy of the development example file to adjust as we go along:

```
$ cp config/development-example.py config/development.py
```

### 3.5.1 Set a Secret Key

A secret key is, among other things, required for login sessions. So let's generate one in a cryptographically secure way:

```
(venv)$ byceps generate-secret-key
3ac1c416bfacb82918d56720d1c3104fd96e8b8d4fbee42343ae7512a9ced293
```

Set this value in your configuration file so the line looks like this:

```
SECRET_KEY = '3ac1c416bfacb82918d56720d1c3104fd96e8b8d4fbee42343ae7512a9ced293'
```

**Attention:** Do **not** use the above key (or any other key you copied from anywhere). Generate your own secret key!

**Attention:** Do **not** use the same key for development and production environments. Generate separate secret keys!

## 3.6 Prepare PostgreSQL

There should already be a system user, likely `postgres`.

Become root:

```
$ su
<enter root password>
```

Switch to the `postgres` user:

```
# su postgres
```

Create a database user named `byceps`:

```
postgres@host$ createuser --echo --pwprompt byceps
```

You should be prompted to enter a password. Do that.

In your *BYCEPS configuration file*, replace the example password in the value of `SQLALCHEMY_DATABASE_URI` with the one you just entered.

Create a schema, also named `byceps`:

```
postgres@host$ createdb --encoding=UTF8 --template=template0 --owner byceps byceps
```

To run the tests (optional), a dedicated user and database have to be created:

```
postgres@host$ createuser --echo --pwprompt byceps_test
postgres@host$ createdb --encoding=UTF8 --template=template0 --owner byceps_test byceps_
↪test
```

Connect to the database:

```
$ psql
```

Load the pgcrypto extension (only necessary on PostgreSQL versions before 13):

```
postgres=# CREATE EXTENSION pgcrypto;
```

Ensure that the function gen_random_uuid() is available now:

```
postgres=# select gen_random_uuid();
```

Expected result (the actual UUID hopefully is different!):

```
           gen_random_uuid
--------------------------------------
 b30bd643-d592-44e2-a256-0e0e167ac762
(1 row)
```

## 3.7 Populate Database

**Important:** Before continuing, make sure that the *virtual environment* is set up and activated.

Create the tables in the database specified in the configuration file:

```
(venv)$ BYCEPS_CONFIG=../config/development.py byceps create-database-tables
Creating database tables ... done.
```

An initial set of authorization roles is provided as a TOML file. Import it into the database:

```
(venv)$ BYCEPS_CONFIG=../config/development.py byceps import-roles scripts/data/roles.
↪toml
Imported 32 roles.
```

With the authorization data in place, create the initial user (which will get all available roles assigned):

```
(venv)$ BYCEPS_CONFIG=../config/development.py byceps create-superuser
Screen name: Flynn
Email address: flynn@flynns-arcade.net
Password:
Creating user "Flynn" ... done.
Enabling user "Flynn" ... done.
Assigning 33 roles to user "Flynn" ... done.
```

Those roles allow the user to log in to the admin backend and to access all administrative functionality.

CHAPTER

FOUR

# RUNNING BYCEPS

## 4.1 Admin Application

**Important:** Before continuing, make sure that the *virtual environment* is set up and activated.

To run the admin application with Flask's (insecure!) *development* server for development purposes:

```
(venv)$ BYCEPS_CONFIG=../config/development.py APP_MODE=admin FLASK_ENV=development␣
↪flask run --debugger --reload
```

The admin application should now be reachable at http://127.0.0.1:5000 (on Flask's standard port).

## 4.2 Site Application

**Important:** Before continuing, make sure that the *virtual environment* is set up and activated.

To run a site application with Flask's (insecure!) *development* server for development purposes on a different port (to avoid conflicting with the admin application):

```
(venv)$ BYCEPS_CONFIG=../config/development.py APP_MODE=site SITE_ID=cozylan FLASK_
↪ENV=development flask run --debugger --reload --port 5001
```

The application for site `cozylan` should now be reachable at http://127.0.0.1:5001.

For now, every site will need its own site application instance.

## 4.3 Worker

**Important:** Before continuing, make sure that the *virtual environment* is set up and activated.

The worker processes background jobs for the admin application and site applications.

To start it:

```
(venv)$ BYCEPS_CONFIG=../config/development.py ./worker.py
```

It should start processing any jobs in the queue right away and will then wait for new jobs to be enqueued.

While technically multiple workers could be employed, a single one is usually sufficient.

# CHAPTER FIVE

# UPGRADING

## 5.1 Python Packages

When updating BYCEPS to a newer version, the set of required Python packages may change (additions, version upgrades/downgrades, removals).

**Important:** Before continuing, make sure that the *virtual environment* is set up and activated.

As with the *installation*, it's probably a good idea to update pip to the current version:

```
(venv)$ pip install --upgrade pip
```

Then instruct pip to install the required Python depdendencies (again, the same way as during the installation):

```
(venv)$ pip install -r requirements.txt
```

This will install new but yet missing packages and upgrade/downgrade existing packages. It will *not* remove no longer used packages, though, but that *should* not be an issue.

If you want to run the test suite and/or use development tools, update their requirements as well:

```
(venv)$ pip install -r requirements-dev.txt
```

CHAPTER

SIX

# LICENSE

BYCEPS is released under the 3-clause BSD license, also known as "New BSD License", "Modified BSD License", and "Revised BSD License".

The license applies both to all of BYCEPS' source code as well as its documentation.

## 6.1 License Text

CHAPTER

# SEVEN

# INDICES AND TABLES

- genindex
- modindex
- search