BYCEPS

Jochen Kupperschmidt

Mar 29, 2024

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.

This documentation should guide you to understand and set up BYCEPS.

- The BYCEPS website is located at https://byceps.nwsnet.de/.
- The source code (including the documentation sources) is available free of charge on GitHub and on Sourcehut.
- If you have questions or suggestions, feel free to reach out on the BYCEPS Discord server.

- If you happen to find an issue or even a bug, please report it on the issue tracker at GitHub.
  - If it could be a severe, security-critical issue, prefer to contact the project author directly.

CHAPTER

ONE

CONCEPTS

1.1 Authorization

User authorization in BYCEPS is based upon permissions and roles.

1.1.1 Structure

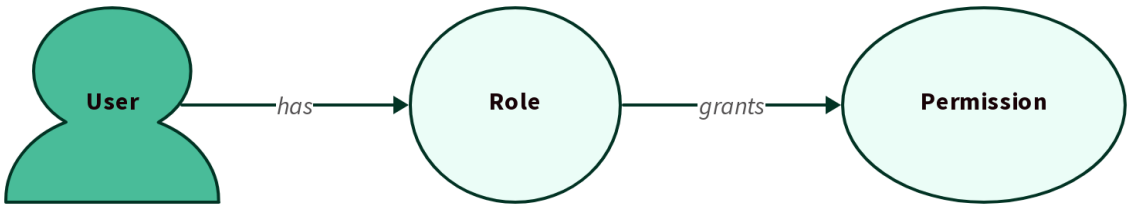

Fig. 1: Relations between entities

- 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.

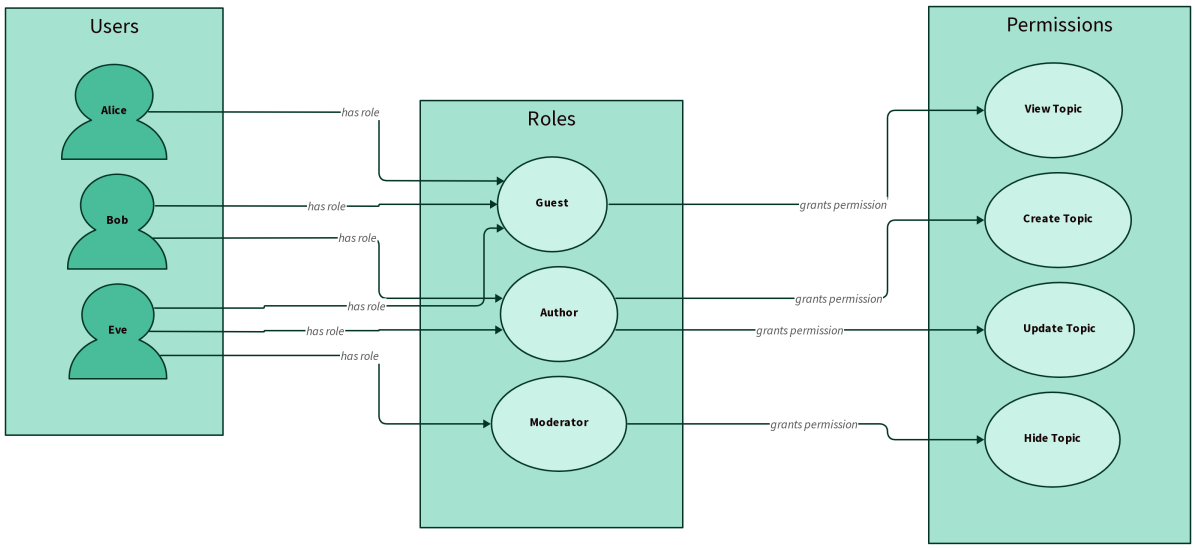

Fig. 2: Example

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

User	Roles	Permissions
Alice	Guest	View Topic
Bob	Guest, Author	View Topic, Create Topic, Update Topic
Eve	Guest, Author, Moderator	View Topic, Create Topic, Update Topic, 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.

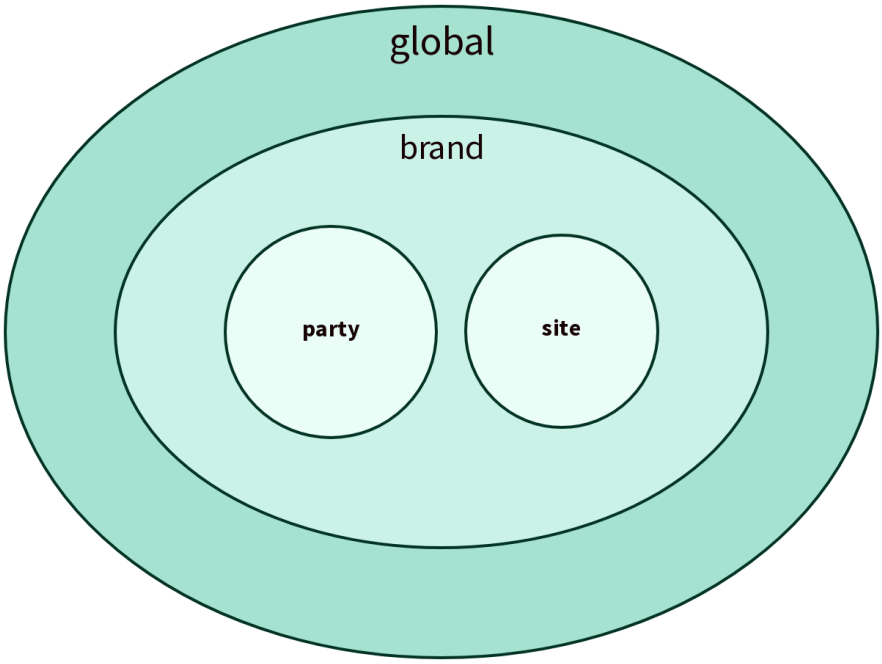

Fig. 3: Nesting of scopes

1.3.1 Global

The global scope is the outermost one.

Entities that belong to the global scope include:

- users
- roles and permissions
- user badges
- *brands*
- global 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:

- email settings
- orga flags
- *parties*
- *sites*
- news channels

- boards
- brand-specific snippets
- terms of service versions

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

1.3.4 Site

The site scope is for entities and settings that belong to a specific website.

Each site belongs to a *brand*.

Entities that belong to the site scope include:

- server name
- pages
- navigation
- site-specific snippets
- choice of
  - news channels
  - forum
  - storefront
  - party
- status of
  - user registration
  - user login

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 BoardTopicCreatedEvent
from byceps.signals.board import topic_created

@topic_created.connect
def celebrate_created_topic(sender, *, event: BoardTopicCreatedEvent = None) -> None:
    print(f'A topic titled "{event.topic_title}" 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

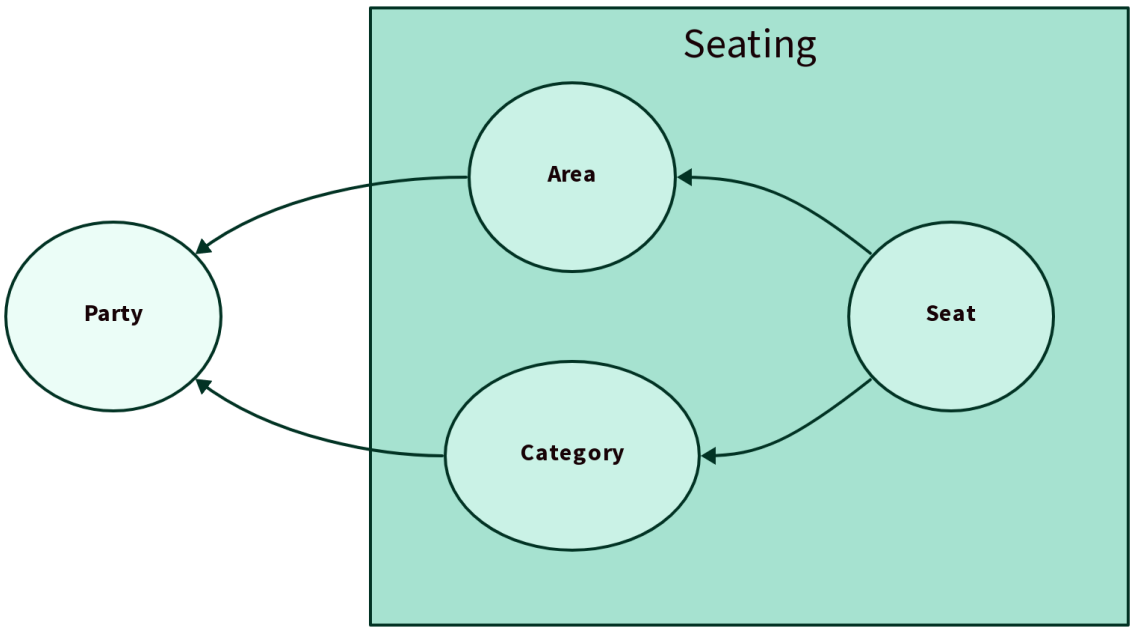

Fig. 1: Relations between entities

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.

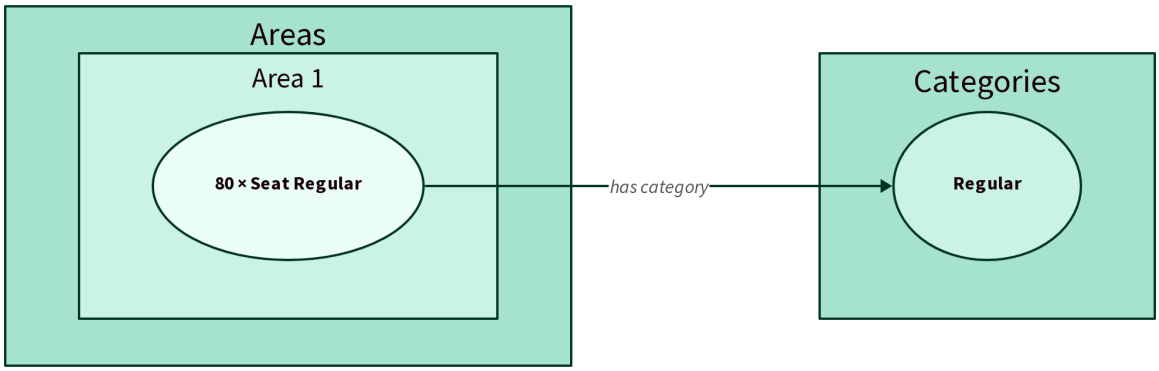

Fig. 2: Small party example

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.

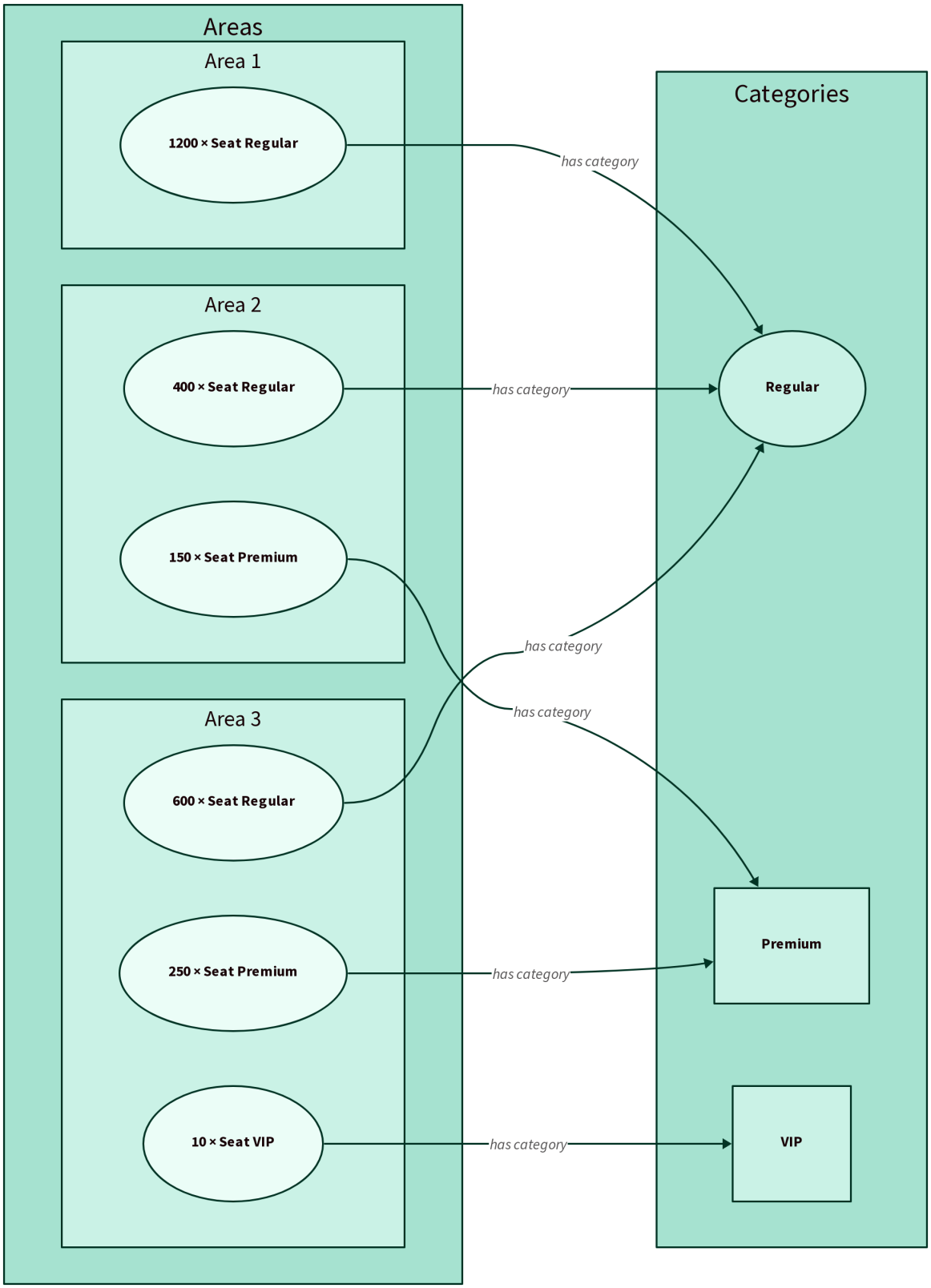

Fig. 3: Big party example

CHAPTER

THREE

INSTALLATION (NATIVE)

This section describes how to install BYCEPS directly on an operating system.

Alternatively, BYCEPS can be run from Docker containers. See *installation with Docker Compose* on how to do that instead.

3.1 Requirements

- A (virtual) server to install BYCEPS on
- At least two subdomains (administration UI, one party website)
- An SMTP server (to send emails)
- Software:
  - Python 3.11 or higher
  - PostgreSQL 13 or higher (for data persistence)
  - Redis 5.0 or higher (for the background job queue)
  - uWSGI, Gunicorn, *or* Waitress (as WSGI server)
  - nginx (as reverse proxy, to serve static files, for TLS)
  - 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 nginx 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 3.11.2
```

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

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

Install the Python dependencies via pip:

```
(venv)$ pip install -r requirements/core.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.toml` and `production_example.toml`, 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.toml config/development.toml
```

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!

3.5.2 Specify SMTP Server

BYCEPS needs to know of an SMTP server, or mail/message transport agent (MTA), to forward outgoing emails to.

The default is to expect a local one on `localhost` and port 25 without authentication or encryption, like Sendmail or Postfix.

Another option is to use an external one (authentication and encryption are important here!) with a configuration like this:

```
MAIL_HOST = "smtp.provider.example"
MAIL_PORT = 465
MAIL_USE_SSL = true
MAIL_USERNAME = "example-username"
MAIL_PASSWORD = "example-password"
```

See the available `MAIL_*` *configuration properties*.

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
```

3.7 Populate Database

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

Initialize the database (*details*) specified in the configuration file:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps initialize-database
Creating database tables ... done.
Importing roles ... done. Imported 35 roles, skipped 0 roles.
Adding language "en" ... done.
Adding language "de" ... done.
```

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

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps create-superuser
Screen name: Flynn
Email address: flynn@flynns-arcade.net
Password:
Creating user "Flynn" ... done.
Enabling user "Flynn" ... done.
Assigning 35 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

INSTALLATION (DOCKER COMPOSE)

As an alternative to *installing directly on a system*, BYCEPS can be run from Docker containers, orchestrated by Docker compose.

Important: This guide assumes you are using Docker Compose V2. If you are still using V1, replace `docker compose` with `docker-compose` before running commands that include it.

Since there is no official Docker image for BYCEPS at this point, you have to build one yourself.

4.1 Obtain BYCEPS

First, clone BYCEPS' Git repository to your machine:

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

A new directory, `byceps`, should have been created. `cd` into it.

4.2 Docker Preparation

Both a `Dockerfile` (to build a Docker image) and a `compose.yml` (to run containers with Docker Compose) come with BYCEPS.

Create the services (build images, create volumes, etc.). This might take a few minutes.

```
$ docker compose up --no-start
```

4.3 Secret Key

Then generate a *secret key* and put it in a file Docker Compose is configured to pick up as a secret:

```
$ docker compose run --rm byceps-apps byceps generate-secret-key > ./secret_key.txt
```

4.4 Database

Now create and initially populate the relational database structure:

```
$ docker compose run --rm byceps-apps byceps initialize-database
```

4.5 Initial User

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

```
$ docker compose run --rm byceps-apps byceps create-superuser
Screen name: Flynn
Email address: flynn@flynns-arcade.net
Password:
Creating user "Flynn" ... done.
Enabling user "Flynn" ... done.
Assigning 35 roles to user "Flynn" ... done.
```

4.6 Hostname-to-Application Routing

Since a single BYCEPS instance can provide the admin frontend, the API, *and* one or more sites, a configuration file is required that defines which hostname will be routed to which application.

Copy the included example configuration file:

```
$ cp config/apps_example.toml config/apps.toml
```

- For a **local installation**, you can go with the examplary hostnames already defined in the example apps configuration file, `config/apps_example.toml`, which are:
  - `admin.byceps.example` for the admin UI
  - `api.byceps.example` for the API
  - `cozylan.example` for the CozyLAN demo site

  To be able to access them, though, add these entries to your local `/etc/hosts` file (or whatever the equivalent of your operating system is):

  ```
  127.0.0.1       admin.byceps.example
  127.0.0.1       api.byceps.example
  127.0.0.1       cozylan.example
  ```

- But if you are **installing to a server**, substitude above hostnames in the config with ones that use actual, registered Internet domains.

4.7 Start BYCEPS

With that configured, spin up the application:

```
$ docker compose up
```

The admin frontend should now be available at http://admin.byceps.example:8080/. Log in with the name of the initial user you created before and the corresponding password.

The "CozyLAN" party site should be accessible at http://cozylan.example:8080/. (If you logged in to the admin frontend just before, you might be logged in already as the same user.)

Attention: For security reasons, BYCEPS only sends cookies back after login over an HTTPS-secured connection by default.

It is expected that BYCEPS is run behind a reverse proxy that adds TLS termination (e.g. nginx or Caddy; often with a certificate from Let's Encrypt).

To be able to login without HTTPS using above links, you can temporarily disable session cookie security by setting *SESSION_COOKIE_SECURE* to false: In `compose.yaml` add `SESSION_COOKIE_SECURE: false` on a separate, indented line to the section `x-byceps-base-env`.

CHAPTER

FIVE

RUNNING BYCEPS

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

5.1 Admin Application

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

```
(venv)$ BYCEPS_CONFIG=../config/development.toml flask --app=serve_admin --debug run
```

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

5.2 Site Application

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.toml SITE_ID=cozylan flask --app=serve_site -
↪-debug run --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.

5.3 Worker

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

To start it:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml ./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

SIX

UPGRADING

6.1 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 dependencies (again, the same way as during the installation):

```
(venv)$ pip install -r requirements/core.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

SEVEN

CONFIGURATION

BYCEPS can be configured with a configuration file. Some values can also be set as environment variables.

7.1 Supported Configuration Values

DEBUG

Enable debug mode.

Default: `False`

Handled by Flask.

Debug mode can also be enabled by appending the `--debug` option to the `flask` command.

DEBUG_TOOLBAR_ENABLED

Enable the debug toolbar (provided by Flask-DebugToolbar).

Default: `False`

JOBS_ASYNC

Makes jobs run asynchronously.

Can be disabled to run jobs synchronously, but that is likely only useful for (and actually used for) testing.

Default: `True`

LOCALE

Specifies the default locale.

Default: `de` (This will likely be changed to `en` at some point in the future.)

MAIL_HOST

The host of the SMTP server.

Default: `'localhost'`

MAIL_PASSWORD

The password to authenticate with against the SMTP server.

Default: `None`

MAIL_PORT

The port of the SMTP server.

Default: `25`

MAIL_STARTTLS

Put the SMTP connection in TLS (Transport Layer Security) mode.

Default: `False`

MAIL_SUPPRESS_SEND

Suppress sending of emails.

Default: `False`

MAIL_USE_SSL

Use SSL for the connection to the SMTP server.

Default: `False`

MAIL_USERNAME

The username to authenticate with against the SMTP server.

Default: `None`

METRICS_ENABLED

Enable the Prometheus-compatible metrics endpoint at `/metrics/`.

Only available on admin application.

Default: `False`

PATH_DATA

Filesystem path for static files (including uploads).

Default: `'./data'` (relative to the BYCEPS root path)

PROPAGATE_EXCEPTIONS

Reraise exceptions instead of letting BYCEPS handle them.

This is useful if an external service like Sentry should handle exceptions.

Default: `None`

If not set, this is implicitly true if `DEBUG` or `TESTING` is enabled.

Handled by Flask.

REDIS_URL

The URL used to connect to Redis.

The format can be one of these:

- `redis://[[username]:[password]]@localhost:6379/0` (TCP socket)
- `rediss://[[username]:[password]]@localhost:6379/0` (SSL-wrapped TCP socket)
- `unix://[[username]:[password]]@/path/to/socket.sock?db=0` (Unix domain socket)

To use the first database of a Redis instance running on localhost on its default port: `redis://127.0.0.1:6379/0`

The documentation for `Redis.from_url` provides details on supported URL schemes and examples.

SECRET_KEY

A secret key that will be for security features such as signing session cookies.

Should be a long, random string.

BYCEPS provides a command-line tool to securely *generate a secret key*.

SESSION_COOKIE_SECURE

Only send cookies marked as secure when an HTTPS connection is available.

Logging in will fail if this is set to true and BYCEPS is accessed without TLS.

This behavior can be disabled for development purposes without a TLS-terminating frontend to the BYCEPS application.

Default: `True` (set by BYCEPS; Flask's default is `False`)

SHOP_ORDER_EXPORT_TIMEZONE

The timezone used for shop order exports.

Default: `'Europe/Berlin'`

SQLALCHEMY_DATABASE_URI

The URL used to connect to the relational database (i.e. PostgreSQL).

Format:

```
postgresql+psycopg://USERNAME:PASSWORD@HOST/DATABASE
```

Example (use user `byceps` with password `hunter2` to connect to database `byceps` on the local host):

```
postgresql+psycopg://byceps:hunter2@127.0.0.1/byceps
```

Since BYCEPS uses psycopg by default, the scheme has to be *postgresql+psycopg*.

For more info, see Flask-SQLAlchemy's documentation on SQLALCHEMY_DATABASE_URI.

SQLALCHEMY_ECHO

Enable echoing of issued SQL queries. Useful for development and debugging.

Default: `False`

STYLE_GUIDE_ENABLED

Enable BYCEPS' style guide, available at `/style_guide/` both in admin mode and site mode.

TESTING

Enable testing mode.

Only relevant when executing tests.

Default: `False`

Handled by Flask.

CHAPTER

EIGHT

COMMAND-LINE INTERFACE

BYCEPS comes with a command-line tool for some tasks.

Important: Before attempting to run any `byceps` command, make sure that the *virtual environment* is set up and activated.

Command	Description
`byceps create-database-tables`	*Create database tables*
`byceps create-superuser`	*Create superuser*
`byceps export-roles`	*Export authorization roles*
`byceps generate-secret-key`	*Generate secret key*
`byceps import-roles`	*Import authorization roles*
`byceps import-seats`	*Import seats*
`byceps import-users`	*Import users*
`byceps initialize-database`	*Initialize database*
`byceps shell`	*Run interactive shell*

8.1 Create Database Tables

`byceps create-database-tables` creates the tables that are required to run BYCEPS in a relational database instance.

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

Note: The *database initialization command* covers this command.

8.2 Import Authorization Roles

`byceps import-roles` imports authorization roles from a file in TOML format into BYCEPS.

By default, an initial set of roles provided with BYCEPS is imported:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps import-roles
Importing roles ... done. Imported 35 roles, skipped 0 roles.
```

Optionally, the file to import from can be specified with the option `-f`/`--file`:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps import-roles -f custom_roles.toml
Importing roles ... done. Imported 35 roles, skipped 0 roles.
```

Note: The *database initialization command* covers this command (except for the option to provide a custom roles file).

8.3 Export Authorization Roles

`byceps export-roles` exports authorization roles in TOML format from BYCEPS to standard output.

To export all roles into a TOML file, standard output is redirected (>) to it:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps export-roles > exported-roles.
↪toml
```

8.4 Initialize Database

`byceps initialize-database` prepares a relational database instance for running BYCEPS.

It is a convenience command that includes the following steps (making it unnecessary to call the covered commands separately):

- Create the database tables. (What *Create Database Tables* does.)
- Import authorization roles. (What *Import Authorization Roles* does.)
- Register the supported languages.

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps initialize-database
Creating database tables ... done.
Importing roles ... done. Imported 35 roles, skipped 0 roles.
Adding language "en" ... done.
Adding language "de" ... done.
```

8.5 Create Superuser

`byceps create-superuser` creates a BYCEPS superuser.

This will:

- create a user account,
- initialize the account,
- assign all existing authorization roles to the account, and
- confirm the associated email address as valid (even though it might not be).

This command is necessary to create the initial user account, which then can be used to log in to the admin backend and to access all administrative functionality.

The command can be run to create additional user accounts as well, but they all will have superuser-like privileges in BYCEPS.

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps create-superuser
Screen name: Flynn
Email address: flynn@flynns-arcade.net
Password:
Creating user "Flynn" ... done.
Enabling user "Flynn" ... done.
Assigning 35 roles to user "Flynn" ... done.
```

Note: This command will only assign the roles that exist in the database. If no roles have been imported, none will be assigned.

8.6 Import Users

`byceps import-users` imports basic user accounts from a file in JSON Lines format into BYCEPS.

This functionality exists to support migration from another system to BYCEPS.

Currently supported fields:

- `screen_name` (required)
- `email_address`
- `legacy_id`
- `first_name`, `last_name`
- `date_of_birth`
- `country`, `zip_code`, `city`, `street`
- `phone_number`
- `internal_comment`

Example file (including a deliberately bad record):

```
{"screen_name": "imported01", "email_address": "imported01@example.test", "first_name":
↪"Alice", "last_name": "Allison"}
{"bad": "data"}
{"screen_name": "imported02", "email_address": "imported02@example.test", "first_name":
↪"Bob", "last_name": "Bobson"}
{"screen_name": "imported03"}
```

To import it:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps import-users example-users.jsonl
[line 1] Imported user imported01.
[line 2] Could not import user: 1 validation error for UserToImport
screen_name
  field required (type=value_error.missing)
[line 3] Imported user imported02.
[line 4] Imported user imported03.
```

8.7 Generate Secret Key

`byceps generate-secret-key` generates a secret key in a cryptographically secure way.

A secret key is, among other things, required for login sessions.

```
(venv)$ byceps generate-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!

8.8 Import Seats

`byceps import-seats` imports seats from a file in JSON Lines format into BYCEPS.

Currently supported fields:

- `area_title` (required)
- `coord_x` (required)
- `coord_y` (required)
- `rotation`
- `category_title` (required)
- `label`
- `type_`

Example file:

```
{"area_title": "Floor 3", "coord_x": 10, "coord_y": 10, "rotation": 0, "category_title":
↪"Premium", "label": "Seat A-1"}
{"area_title": "Floor 3", "coord_x": 25, "coord_y": 10, "rotation": 0, "category_title":
↪"Premium", "label": "Seat A-2"}
```

To import it:

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps import-seats my-party-2023␣
↪example-seats.jsonl
[line 1] Imported seat (area="Floor 3", x=10, y=10, category="Premium").
[line 2] Imported seat (area="Floor 3", x=25, y=10, category="Premium").
```

8.9 Run Interactive Shell

The BYCEPS shell is an interactive Python command line prompt that provides access to BYCEPS' functionality as well as the persisted data.

This can be helpful to inspect and manipulate the application's data by using primarily the various services (from `byceps.services`) without directly accessing the database (hopefully limiting the amount of accidental damage).

```
(venv)$ BYCEPS_CONFIG=../config/development.toml byceps shell
Welcome to the interactive BYCEPS shell on Python 3.11.2!
>>>
```

CHAPTER

NINE

TESTING

BYCEPS comes with a quite extensive (but not all-encompassing) suite of tests to be able to verify that at least a big part works as intended.

Running the tests is mostly useful for development of BYCEPS itself as well as for customization.

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

In the activated virtual environment, first install the test dependencies:

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

Then run the tests:

```
(venv)$ pytest
```

To abort on encountering the first failing test case:

```
(venv)$ pytest -x
```

CHAPTER

TEN

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.

10.1 License Text

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX