# Work Package for Applicants

***Release 0.0***

**root**

**Jan 14, 2020**

## Quickstart

Use the definitions from table 1 for replacements in commands:

table 1: Placeholders and replacements

| Placeholder | Replacement |
|---|---|
| @user-index@ | 1, 2, . . . |
| @offset@ | 50 + @user-index@, i.e. 51, 52, . . . |
| @user@ | applicant@user-index@ |
| @passwd@ | see VNC password in README-vnc.html |

**Abstract**

Task specification for building a minimal Python web application using specific tools and build environment.

The emphasis of this test is on documentation and review workflow, the functionality is secondary.

You are expected to complete the project

- with an incomplete draft specification (the required documentation is both design and user manual)
- incrementally in an environment with a version control system (Mercurial) and review tool (Kallithea)
- in a design/review/implementation/review cycle
- according to defined procedural standards, which are not optional (i.e. consult the provided documentation)

# Contents

# List of Figures

# List of Tables

# List of Code Blocks

CHAPTER 1

# Introduction

The development environment is

- Linux (ubuntu)
- emacs(1) (with home-grown extensions)
- snippets(1) tool (templating)
- terminal/shell/make
- Python 2.x/3.x
- Mercurial (TortoiseHg thg(1) is installed as GUI)
- Kallithea at http://sw-amt.ws/kallithea/applicants/@user@

Although it is possible to use editors other than emacs for the current task, the missing highlighting support and tag navigation make comprehension and work quite hard.

The task can be completed in a terminal with ssh(1) access to *sw-amt.ws*. It is perfectly possible to use PuTTY under windows.

A more comfortable graphical environment is available with a VNC viewer (e.g. UltraVNC) via SSH-Tunnel. MacOS X comes with a builtin VNC client, press ⌘-k (CMD-k) in Finder and enter *vnc://host:port*.

Since the project is under version control, it is also possible to clone the repository and complete the project entirely off-site.

CHAPTER 2

# Task outline

Access *sw-amt.ws* with ssh(1) in terminal.

1. Initialize `workpackage` directory from repository:

```
hg clone http://@user@:@passwd@@sw-amt.ws/kallithea/applicants/@user@ workpackage
```

2. Initialize documentation in `workpackage` directory:

```
cd workpackage
sda init Workpackage
```

   Commit initial revision to version control system (VCS):

```
hg add
hg commit -m 'README.txt: project documenatation initial revision'
hg push
```

3. Create database `workpackage.db`:

```
snn --key sqlite_db_base --value workpackage workpackage.sql
emacs workpackage.sql
# create table ...
sqlite3 workpackage.db <workpackage.sql
```

   Commit initial revision to VCS:

```
hg add workpackage.sql
hg commit -m 'workpackage.sql: define tables'
hg push
```

4. Generate `workpackage_db` module:

```
sqa-schema.py --pyjsmo-schema 'sqlite:///workpackage.db' > 'workpackage_db.py'
emacs 'workpackage_db.py'
```

   Fill out variable DSN with *workpackage.db*.
   Analyze test/example code at end of file.

   Commit initial revision to VCS:

```
hg add workpackage_db.py
hg commit -m 'workpackage_db.py: define sqlalchemy ORM tables'
hg push
```

**Note:** Use `sqlalchemy` ORM classes from `workpackage_db` to access the database tables. **Do not use low-level SQL statements** based on PEP 249 – Python Database API Specification v2.0.

5. Build and view documentation:

```
sda make
xdg-open https://sw-amt.ws/@user@/doc/html
```

Accessing the generated documentation at https://sw-amt.ws/@user@/doc/html requires the username @*user*@ and the same password as for the VNC server.

Examples for diagrams and pointers to definitions can be found in Documentation Standard.

**Note:** The documentation ist embedded into a framework based on Sphinx. There is no need to use any facilities provided by the framework. The documentation goes between the lines

```
.. >>CODD Introduction

-- DOCUMENTATION GOES HERE --

.. >>CODD Notes
```

6. Design `workpackage` module in `README.txt`.

Create bottle app. In function `run()` start HTTP server on port supplied with option –port, default: 50@offset@.

On index page display buttons *Fill* and *Clear*. Also display all persons and their car(s) in a table.

- *Fill* => fill database with sample records, display index
- *Clear* => clear database, display index

Use PlantUML UML diagrams for design:

- class diagram for class *person* and class *car* with relations
- activity diagram for functions

Commit changes into VCS:

```
hg commit -m 'README.txt: program design'
hg push
```

Submit for review.

7. Implement `workpackage` module:

```
snn workpackage.py
emacs 'workpackage.py'
```

Commit initial revision and meaningful increments to VCS.

**Note:** missing python packages can be installed in a *virtualenv* or with `pip install --user`, e.g.:

```
pip install --user bottle
```

8. Serve `workpackage` application and view it:

```
python workpackage.py
xdg-open https://sw-amt.ws/@user@/persons
```

The URL https://sw-amt.ws/@user@/persons is served by Apache with a reverse proxy configuration to http://localhost:50@offset@

9. Complete documentation of project *workpackage*:

```
emacs 'README.txt'
```

# CHAPTER 3

## Task details

1. Create a *sqlite3* database from an SQL file `workpackage.sql`, with tables *person* (see table 3.1) and *car* (see table 3.2):

   **Note:** Do not declare the foreign key relation for *driver_id* explicitely.

table 3.1: person

| column | type | attributes |
|---|---|---|
| _id | INTEGER | AUTOINCREMENT, PRIMARY KEY |
| name | TEXT | |
| first_name | TEXT | |

table 3.2: car

| column | type | attributes |
|---|---|---|
| _id | INTEGER | AUTOINCREMENT, PRIMARY KEY |
| manufacturer | TEXT | |
| model | TEXT | |
| driver_id | INTEGER | |

CHAPTER 4

# Administration

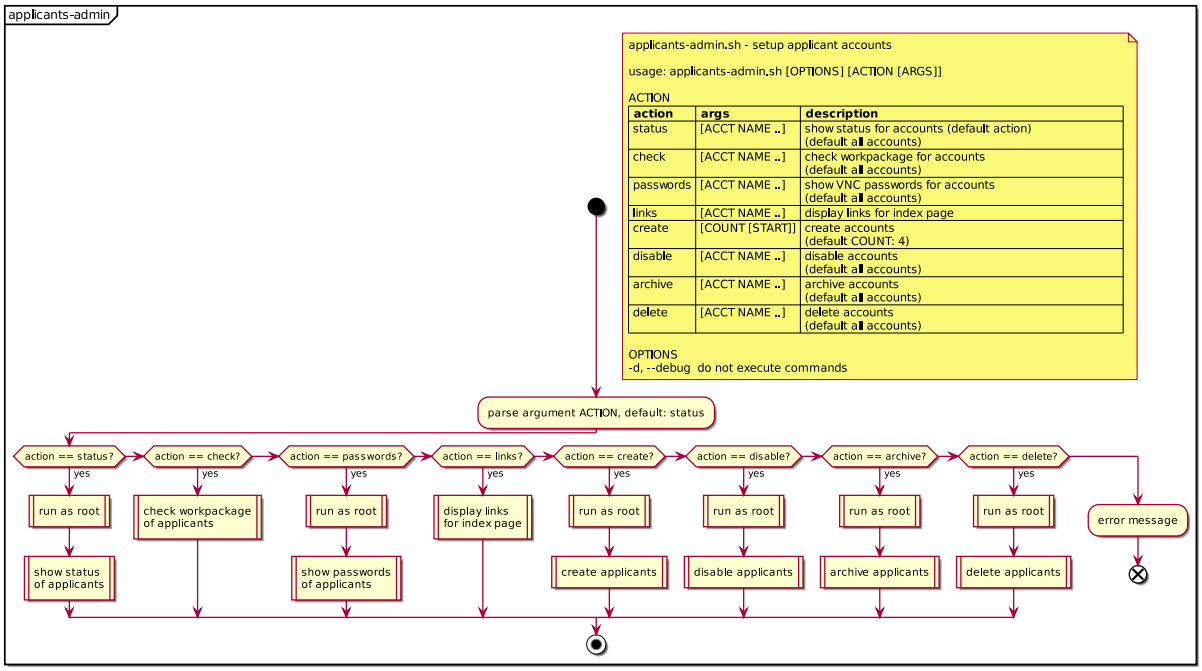
The script **`applicants-admin.sh`** provides an administration shell to manage applicant accounts.

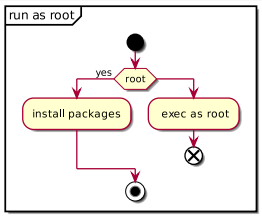
## 4.1 Diagrams

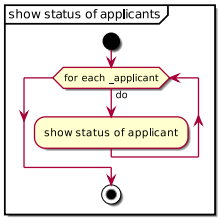
(see figure 4.1)

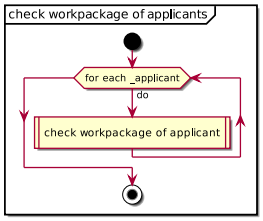
### 4.1.1 Functions

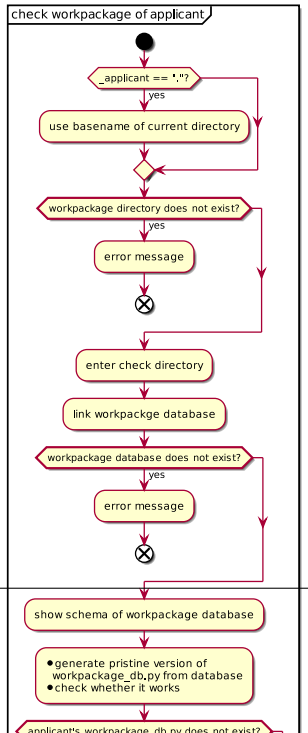
(see figure 4.2)


applicants-admin
applicants-admin.sh - setup applicant accounts
usage: applicants-admin.sh [OPTIONS] [ACTION [ARGS]]
ACTION
action | args | description
status | [ACCT NAME ..] | show status for accounts (default action) (default all accounts)
check | [ACCT NAME ..] | check workpackage for accounts (default all accounts)
passwords | [ACCT NAME ..] | show VNC passwords for accounts (default all accounts)
links | [ACCT NAME ..] | display links for index page
create | [COUNT [START]] | create accounts (default COUNT: 4)
disable | [ACCT NAME ..] | disable accounts (default all accounts)
archive | [ACCT NAME ..] | archive accounts (default all accounts)
delete | [ACCT NAME ..] | delete accounts (default all accounts)
OPTIONS
-d, --debug do not execute commands
parse argument ACTION, default: status
action == status?
action == check?
action == passwords?
action == links?
action == create?
action == disable?
action == archive?
action == delete?
yes
run as root
show status of applicants
check workpackage of applicants
run as root
show passwords of applicants
display links for index page
run as root
create applicants
run as root
disable applicants
run as root
archive applicants
run as root
delete applicants
error message


run as root
yes
root
install packages
exec as root


show status of applicants
for each _applicant
do
show status of applicant


check workpackage of applicants
for each _applicant
do
check workpackage of applicant


check workpackage of applicant
_applicant == "."?
yes
use basename of current directory
workpackage directory does not exist?
yes
error message
enter check directory
link workpackge database
workpackage database does not exist?
yes
error message
show schema of workpackage database
generate pristine version of workpackage_db.py from database
check whether it works

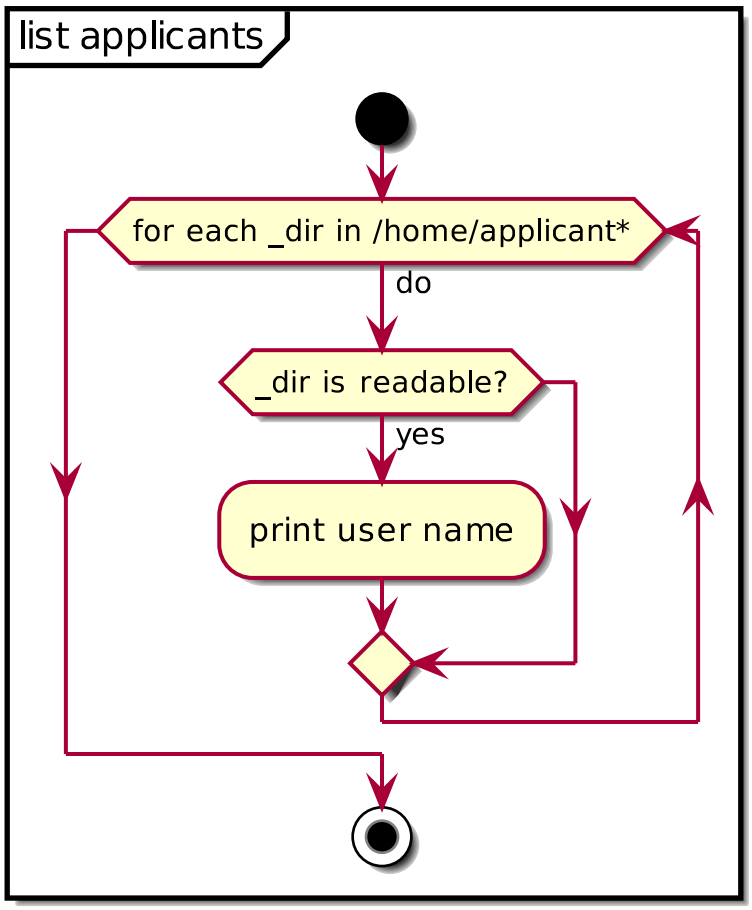



figure 4.2: List all applicants