数据库及接口设计

Models

Tables

- administrator
  - id
  - username
  - pwhash1
  - token
  - is_root
  - name
  - contact_info
- school
  - id
  - name
  - person_in_charge ⟶ administrator.id
- building
  - id
  - school_id ⟶ school.id
  - name
  - person_in_charge ⟶ administrator.id
- session
  - id
- cat1
  - id
  - name
- cat2
  - id
  - cat1_id ⟶ cat1.id
  - name
- file

    - id
    - filename

- product
    - id
    - name
    - cat2_id ⟶ cat2.id
    - pic_id ⟶ file.id
    - description
    - price

- product_building
    - product_id ⟶ product.id
    - building_id ⟶ building.id
    - quantity

- cart
    - session_id ⟶ session.id
    - product_id ⟶ product.id
    - quantity

- order
    - id
    - session_id ⟶ session.id
    - building_id ⟶ building.id
    - room
    - receiver
    - phone
    - status (uncommitted, uncompleted, completed, cancelled)
    - eta
    - updated_time
    - password

- order_product
    - order_id ⟶ order.id
    - product_id ⟶ product.id
    - quantity

- promotion
    - id
    - pic_id ⟶ file.id

Views or materialized views (for performance & convenience concerns,

optional)

The following view definitions are untested and just as a reference.

```
CREATE VIEW product_list AS
SELECT id, building_id, name,
  cat1_id, cat1.name AS cat1_name, cat2_id, cat2.name AS cat2_name,
  pic_id, description, price, quantity
FROM product
JOIN product_building ON product.id = product_building.product_id
LEFT OUTER JOIN cat2 ON product.cat2_id = cat2.id
LEFT OUTER JOIN cat1 ON cat2.cat1_id = cat1.id

CREATE VIEW order_detailed_list AS
SELECT id, session_id, room, receiver, phone, building_id,
  building.name AS building_name, school.name AS school_name,
  status, eta, updated_time, password,
  product_id, product.name AS product_name, price, quantity, price * quantity AS total
FROM order
JOIN building ON order.building_id = building.id
JOIN school ON building.school_id = school.id
JOIN order_product ON order.id = order_product.order_id
JOIN product ON order_product.product_id = product.id

CREATE VIEW cart_detail AS
SELECT session_id, building.id as building_id,
  product.id AS product_id, product.name AS product_name, price,
  cart.quantity AS quantity, product_building.quantity AS quantity_avail
FROM cart
JOIN product ON cart.product_id = product.id
JOIN session ON cart.session_id = session.id
JOIN product_building ON
  session.building_id = product_building.building_id AND
  cart.product_id = product_building.product_id
```

Views / APIs

- `GET /`

  Main page.

- `POST /choose_location`

  Choose the user's current location. Form argument:

  - `building_id`

After choosing the building, the back-end should create a new session, and store the session id, building id and any other necessary information into Cookies, and than redirect to a specified page (the main page or previous page).

- `GET /products/<cat1_id:int>/<cat2_id:int>`

  Product list under a specified category. `cat2_id` is optional and if it is not under `cat1`, emit an error.

- `GET /products/details/<product_id:int>`

  Product details page for `product_id`.

- `GET /cart`

  View the cart.

- `POST /cart/add`

  Add a product to the cart. Form arguments:

  - `product_id`
  - `quantity`

- `POST /cart/edit`

  Modify an item in the cart. Form arguments:

  - `product_id`
  - `quantity` Remove the item in the cart if `0`.

- `GET /order/counter`

  Go to the counter. The user should provide the receiver's information in this page. The user can modified that information in this page as well when coming back from the order review page.

- `POST /order/counter`

  Create an order of the products in the current cart, along with the receiver's information provided in the submitted form. Form arguments:

  - `room`
  - `receiver`
  - `phone`

  Other fields in `order` should be initialized at the same time. The order id should be store in the session (the *session* here doesn't mean the table described above but the concept in Web

development). Redirect to `/order/view` when finished.

- `GET /order/review`

  Review the order created just now (specified by the order id in the session). May go back to the counter.

- `POST /order/submit`

  Submit the order. Change the order status from `uncommitted` to `uncompleted`.

- `GET /order/list`

  View the order list of the current user.

- `GET /admin`

  Admin main page. Show different information or redirect to different pages based on the administrator's privilege.

- `GET/POST /admin/login`

  Administrator login. Back-end should provide a random salt for front-end and store the *encrypted or signed* salt in the session (which should be already done automatically by flask's builtin session facility).Form arguments:

  - `username`
  - `pwhash2`

  Password hashing function for reference:

  ```
  pwhash1 = PBKDF2(key=password, salt=username)
  pwhash2 = PBKDF2(key=pwhash1, salt=back_end_specifing_salt)
  ```

- `POST /admin/logout`

  Logout.

- `POST /admin/change_password`

  Change password. Form arguments:

  - `username`
  - `old_pwhash2` Must matches the old password.
  - `new_pwhash1`

- `GET /admin/order/<building_id:int>`

Order management page of the building specified by `building_id`.

- `GET /admin/order/details/<order_id:int>`

  View the order details.

- `POST /admin/order/change_status/<order_id:int>`

  Change the order status. The updated time of the order should be changed as well. Form arguments:

  - `new_status`
  - `password` May not be needed with some administration privileges. It depends on specific business logic.

- `GET /admin/inventory/<building_id:int>`

  Inventory management page of the building specified by `building_id`.

- `POST /admin/inventory/<building_id:int>/edit`

  Edit the inventory information of a building. Modify a single record each time. Create a new record in `product_building` as needed. Form arguments:

  - `product_id`
  - `quantity`

- `GET /admin/product`

  Product information management.

- `POST /admin/product/add`

  Create a new product. Form arguments:

  - `name`
  - `cat2_id`
  - `pic_id`
  - `description`
  - `price`

- `POST /admin/product/edit/<product_id:int>`

  Modify the information of a product. Form arguments:

  - `name`
  - `cat2_id`
  - `pic_id`

    - `description`
    - `price`

- `POST /admin/product/delete/<product_id:int>`

  Delete a product.

- `GET /admin/cat1`

  Top level product category management.

- `POST /admin/cat1/add`

  Add a top level product category. Form argument:

    - `name`

- `POST /admin/cat1/edit/<cat1_id:int>`

  Modify a top level category. Form argument:

    - `name`

- `POST /admin/cat1/delete/<cat1_id:int>`

  Delete a top level category.

- `GET /admin/cat1/<cat1_id:int>/cat2`

  Second level product category under `cat1`.

- `POST /admin/cat1/<cat1_id:int>/cat2/add`

  Add a second level category under `cat1`. Form argument:

    - `name`

- `POST /admin/cat1/<cat1_id:int>/cat2/edit/<cat2_id:int>`

  Edit a second level category. Form argument:

    - `name`

- `POST /admin/cat1/<cat1_id:int>/cat2/delete/<cat2_id:int>`

  Delete a second level category.

- `GET /admin/file`

File (attachment) management.

- `POST /admin/file/upload`

  Upload a file. Form arguments (multipart form format):

  - `filename`
  - `file`

- `POST /admin/file/remove/<file_id:int>`

  Delete a file.

- `GET /admin/promotion`

  Promotion event management.

- `POST /admin/promotion/add`

  Add a new promotion event. Form argument:

  - `pic_id`

- `POST /admin/promotion/delete/<promotion_id:int>`

  Delete a promotion.

- `GET /admin/administrator`

  Administrator management page.

- `POST /admin/administrator/add`

  Add an administrator. Form arguments:

  - `username`
  - `pwhash1`
  - `is_root`
  - `name`
  - `contact_info`

- `POST /admin/administrator/edit`

  Modify the information of an administrator. Form arguments:

  - `username`
  - `pwhash1` Unchanged if `null`.
  - `is_root`

    - `name`
    - `contact_info`
- `POST /admin/administrator/delete`

  Delete an administrator. Form argument:

    - `username`
- `GET /admin/school`

  School management page.
- `POST /admin/school/add`

  Add a new school. Form arguments:

    - `name`
    - `person_in_charge`
- `POST /admin/school/edit/<school_id:int>`

  Modify the information of a school. Form arguments:

    - `name`
    - `person_in_charge`
- `POST /admin/school/delete/<school_id:int>`

  Delete a school.
- `GET /admin/school/<school_id:int>/building`

  Building management of the school specified by `school_id`.
- `POST /admin/school/<school_id:int>/building/add`

  Add a building to a school. Form arguments:

    - `name`
    - `person_in_charge`
- `POST /admin/school/<school_id:int>/building/edit/<building_id:int>`

  Modify the information of a building. Form arguments:

    - `name`
    - `person_in_charge`

- `POST /admin/school/<school_id:int>/building/delete/<building_id:int>`

  Delete a building.