CSC 648-848 section 01
Milestone 3 on-line review template

Fall 2018

****** Please fill out this template, convert it to PDF and mail as attachment to Petkovic@sfsu.edu by 11/16/18 noon. You will get feedback soon after. Anthony will do off-line review as per M3 Appendix II template ******

In subject line of your e-mail put "CSC 648-848 01 M3 online review Team N"

Team: Section 01_Team 02

Date: 16 Nov 2018

1. Please briefly answer each question below:

a) *Teamwork*: is the team working out, any related issues

Teamwork is good. The team works well together, and we have good communication between the backend and frontend team. Our team lead takes care that everyone is on the same page. In short, there are no issues going around with the teamwork.

b) *Risks*: list all actual (not hypothetical) risks (schedule, technical, skills etc.) and plans how to resolve them OR ask for help.

We are using file system for media storage. Using file system, the link to the image is stored in the database. Thumbnails are created on upload of the image. However, we are facing some challenges as we deployed the website on Heroku yesterday. In our website, we upload an image for posting an item, we did that and could see it in the "All Categories" page. However, today the images disappeared. Everything else seems to work fine. The application is working fine on our local machine as well as on Heroku, other than the image which is not there.

Upon searching, we came up with the following reasons on stack Overflow for this, Heroku does not support file uploads. The filesystem is read only. Heroku file system is temporary and not shared across dynos or restarts - which Heroku does every day. Hence, the images may get disappeared. As a temporary workaround, we are hosting the files elsewhere as of now.

The plan to resolve this issue will be to use DB BLOBS. The team does not have enough knowledge about this, but we will try to learn this over this weekend. Also, if you can provide any guidance on it, that will be very helpful for us.

c) *Coding practices*: Have you decided which coding style to use and actually use it? Make sure you sue header comments and some code inline comments too.

Yes, we followed the below coding practices. Also, we referred to our class notes and that was very helpful.

- Comment conventions
- Indent style conventions
- Line length conventions
- Naming conventions
- Programming practices
- Programming style conventions
- Meaningful branch names on GitHub
- Meaningful commit comments on GitHub

d) *Usage of proper SE code management practices in GitHub*

Do you have master branch that contain only tested code and is carefully managed?

Yes, we do have a master branch that contains only tested code and is managed by the git master and team lead.

Do you have development and feature branches?

Yes, we do have different development and feature branches on GitHub.

Are you doing any code review?

Yes, code review is being done by our git master.

Are your comments on GitHub code commits meaningful to reflect enough details?

Initially, for some comments we did not have meaningful git commits. But now, we are making sure to enter meaningful git comments before every commit.

Are all team members committing on GitHub individually?

Yes, all the team members are committing on GitHub individually for their respective work

e) *How did you address site security and safe coding practices?*

Are PW encrypted/hashed

Yes, the PW is encrypted.

Are you doing any input validation on search text entry as specified (<40 alphanumeric characters?

Yes, input validation on search text entry as specified has been taken care of.

f) *Digital content* (e.g. images, video) – have you obtained enough (total of 30 or so) images for e.g. 4-5 product categories for your demo, royalty free

Yes, we have enough images for our demo and we have taken these images from google images by selecting 'free to use or share' in advanced settings.

g) Anything else
Nothing specific as of now.

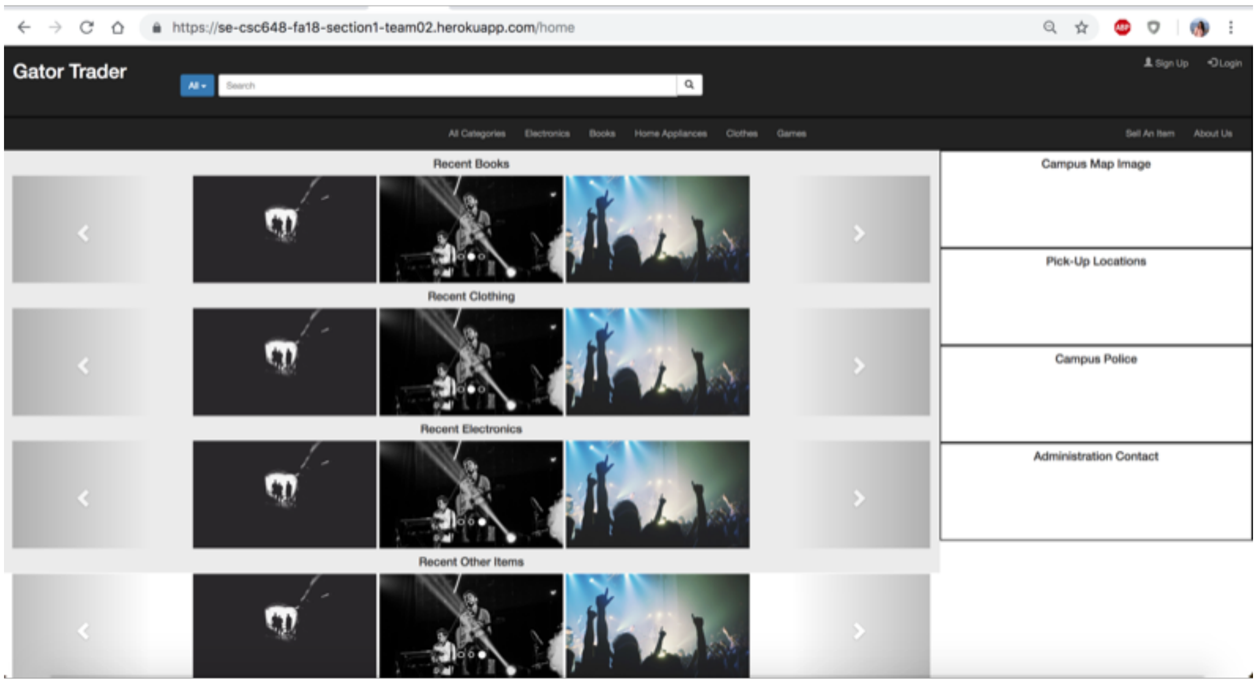
2. Provide URL to home page, running from your server so it can be tested (ensure UI implementation is responsive)

URL: **https://se-csc648-fa18-section1-team02.herokuapp.com**

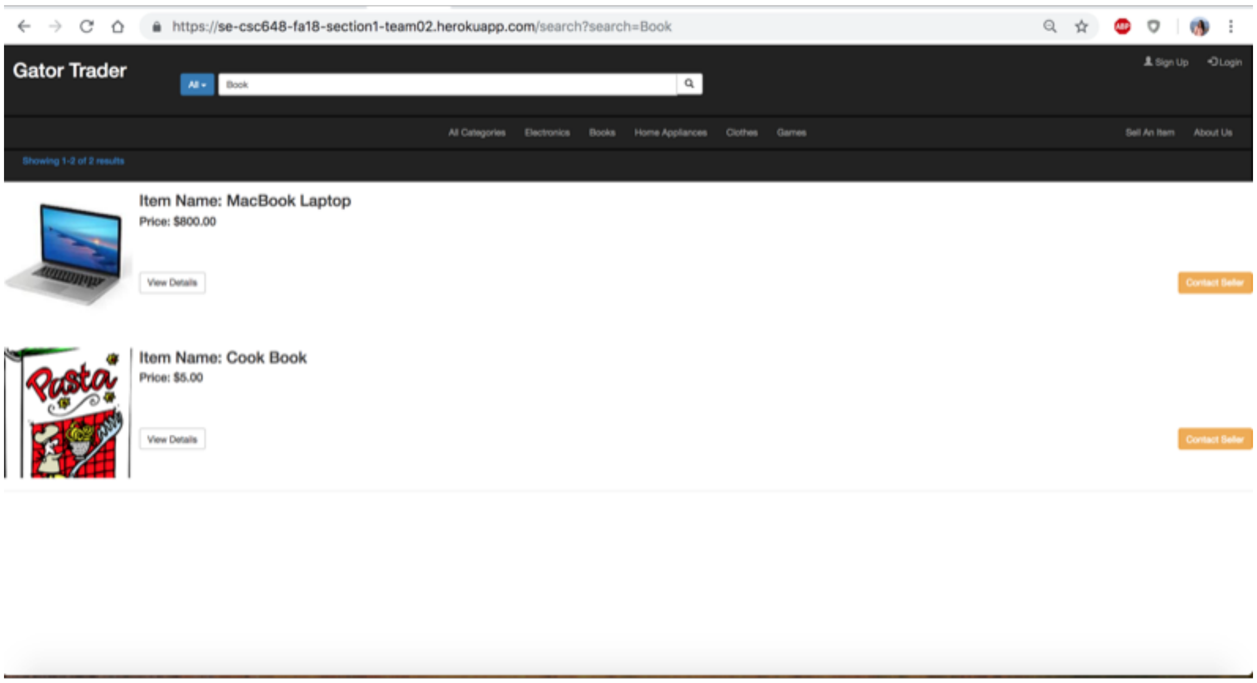
Working pages - Home page, Search Page, Log In page, Sign Up Page, Sell an Item page, About Us

3. Provide PDF screen shots, ONE PER PAGE, of the following screens (they have to be real coded pages but not necessarily connected to back end)

- Home page



- Search results (NOTE: search parameters chosen by user have to be persistent)

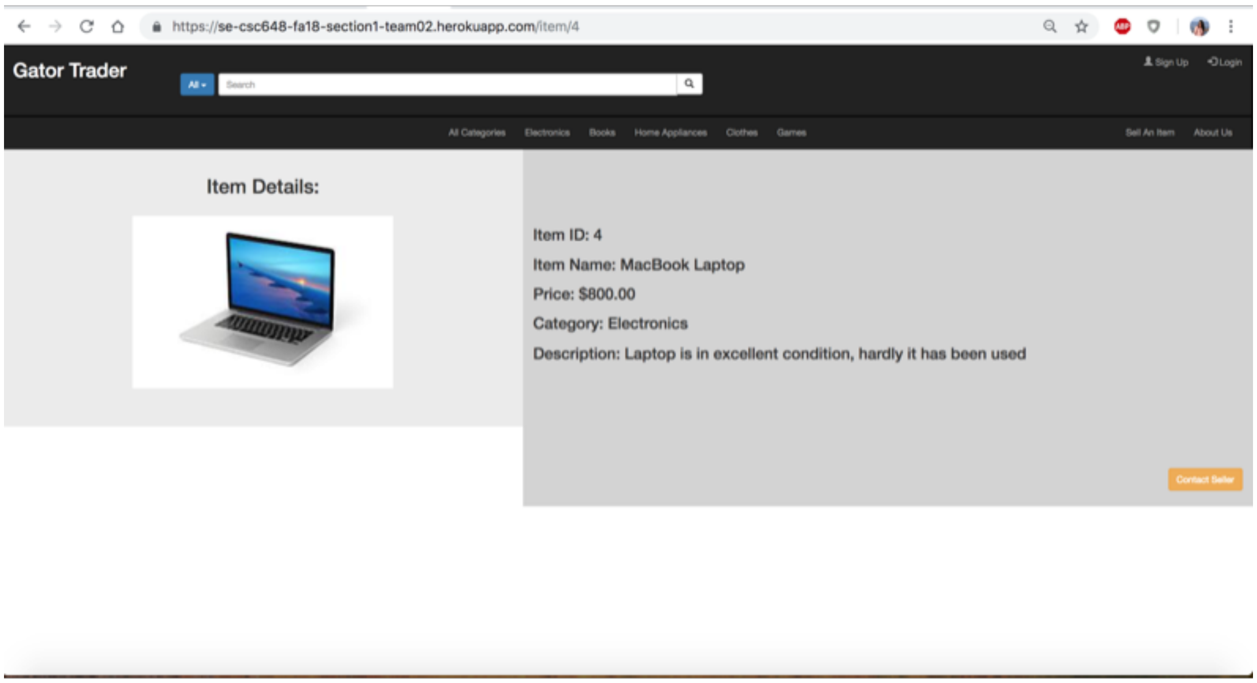

https://se-csc648-fa18-section1-team02.herokuapp.com/search?search=Book

Showing 1-2 of 2 results

Item Name: MacBook Laptop
Price: $800.00

Item Name: Cook Book
Price: $5.00

- Item details



Item Details:

Item ID: 4

Item Name: MacBook Laptop

Price: $800.00

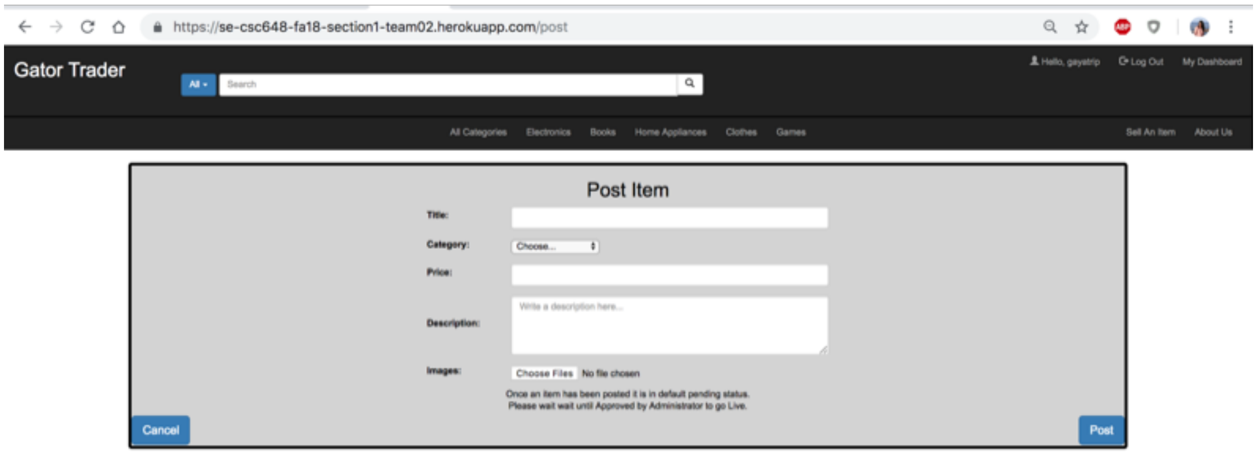
Category: Electronics

Description: Laptop is in excellent condition, hardly it has been used

- Messaging/contact seller

As of now, we just have the button implementation to this as shown in search page and Item details page.

- Item posting



https://se-csc648-fa18-section1-team02.herokuapp.com/post

Post Item

Title:

Category: Choose...

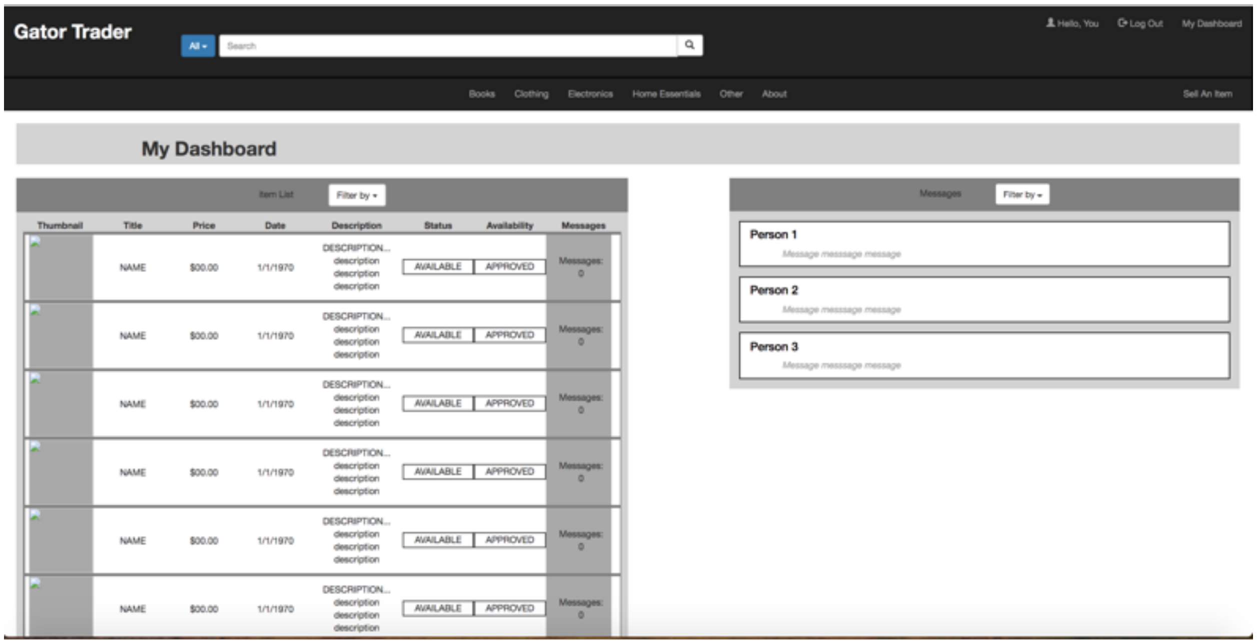
Price:

Description: Write a description here...

Images: Choose Files No file chosen

Once an item has been posted it is in default pending status.
Please wait wait until Approved by Administrator to go Live.

- Seller dashboard



My Dashboard

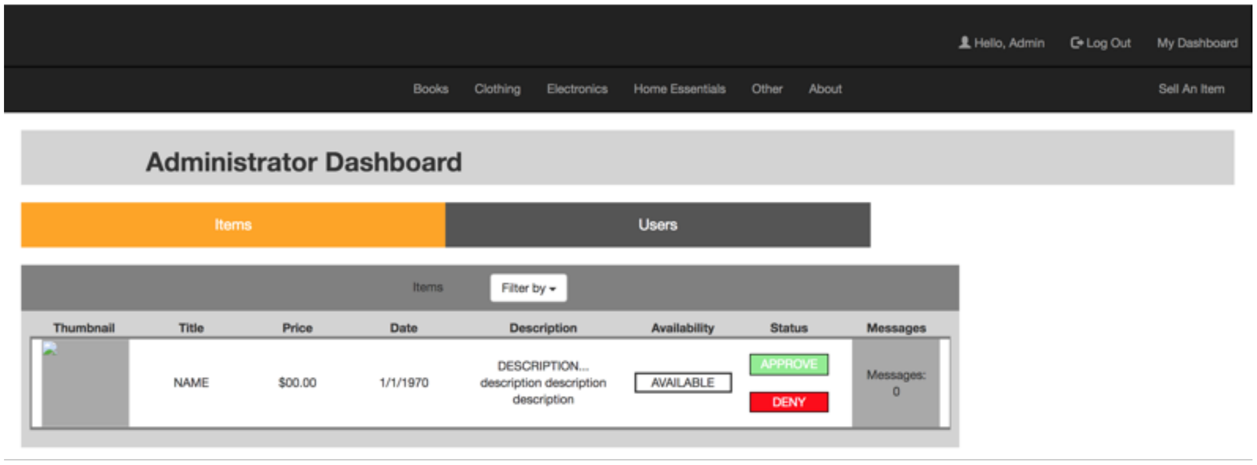
Thumbnail	Title	Price	Date	Description	Status	Availability	Messages
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVED	Messages: 0
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVED	Messages: 0
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVED	Messages: 0
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVED	Messages: 0
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVED	Messages: 0
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVED	Messages: 0

Person 1
Message messsage message

Person 2
Message messsage message

Person 3
Message messsage message

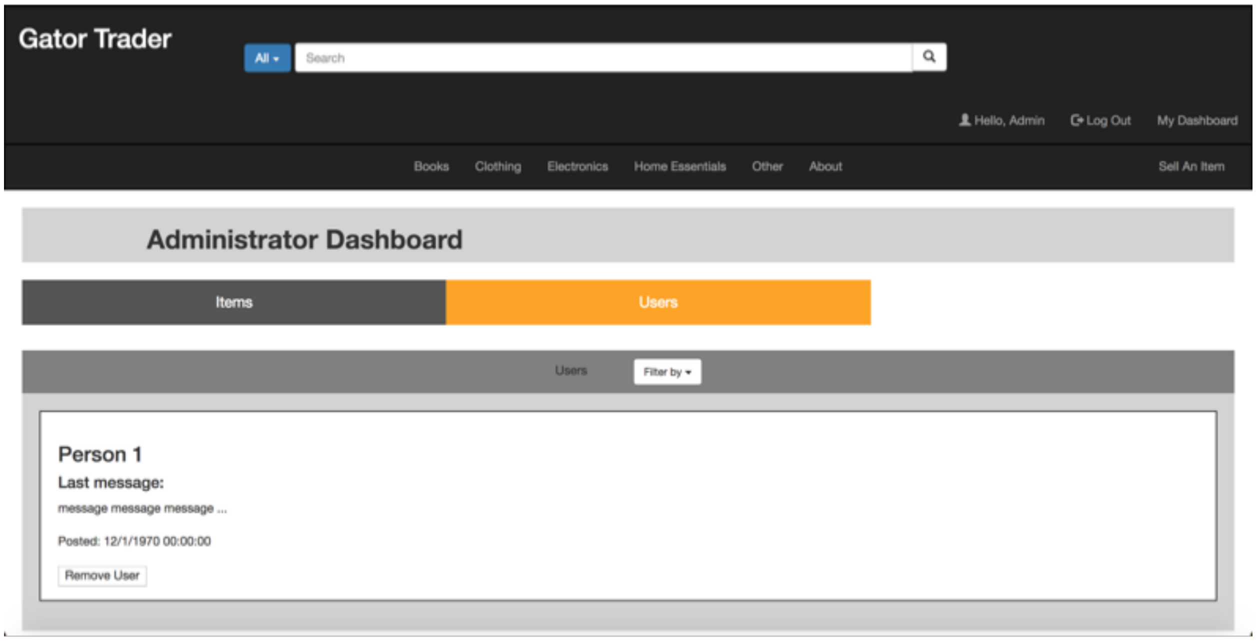
- Admin dashboard
    1. Item dashboard



Administrator Dashboard

Thumbnail	Title	Price	Date	Description	Availability	Status	Messages
	NAME	$00.00	1/1/1970	DESCRIPTION... description description description	AVAILABLE	APPROVE DENY	Messages: 0

2. User dashboard



Person 1

Last message:

message message message ...

Posted: 12/1/1970 00:00:00

4. List all major DB tables you have implemented and list their items

The database tables are as below: -

4.1.1 user_record - This is the user registration table when an unregistered user registers on Gator Trader.

```
CREATE TABLE public.user_record
(
    user_id integer NOT NULL DEFAULT nextval('user_record_user_id_seq'::regclass),
    user_name text COLLATE pg_catalog."default" NOT NULL,
    user_password text COLLATE pg_catalog."default" NOT NULL,
    user_email text COLLATE pg_catalog."default" NOT NULL,
    admin_right boolean NOT NULL,
    user_date timestamp(6) with time zone DEFAULT now(),
    CONSTRAINT user_record_pkey PRIMARY KEY (user_id),
    CONSTRAINT user_record_user_email_key UNIQUE (user_email)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

4.1.2 item - This is the item table which creates the item data when a registered user posts an item for sale.

```
CREATE TABLE public.item
(
    item_id integer NOT NULL DEFAULT nextval('item_item_id_seq'::regclass),
    item_title text COLLATE pg_catalog."default" NOT NULL,
    item_description text COLLATE pg_catalog."default" NOT NULL,
    item_price money NOT NULL,
    item_status text COLLATE pg_catalog."default" NOT NULL,
    user_id integer NOT NULL DEFAULT nextval('item_user_id_seq'::regclass),
    category_id integer NOT NULL DEFAULT nextval('item_category_id_seq'::regclass),
    item_image text COLLATE pg_catalog."default" NOT NULL,
    item_availability boolean NOT NULL,
    item_image_thumbnail text COLLATE pg_catalog."default" NOT NULL,
    item_date timestamp(6) with time zone DEFAULT now(),
    CONSTRAINT item_pkey PRIMARY KEY (item_id),
    CONSTRAINT fk_category FOREIGN KEY (category_id)
        REFERENCES public.category (category_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_user FOREIGN KEY (user_id)
        REFERENCES public.user_record (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

4.1.3 category - This table categorizes item. Whenever a registered user posts an item he has to select the category for an item. Category selection would be mandatory for sellers and they would not be allowed to enter his choice for category. It will have the following data fields.

```
CREATE TABLE public.category
(
    category_id integer NOT NULL DEFAULT
    nextval('category_category_id_seq'::regclass),
    category_name text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT category_pkey PRIMARY KEY (category_id)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

4.1.4 gator_message - This table contains the messages from registered users who are interested in buying the items posted for sale. It will have the following data fields.

```
CREATE TABLE public.gator_message
(
    message_id integer NOT NULL DEFAULT
nextval('gator_message_message_id_seq'::regclass),
    message_text text COLLATE pg_catalog."default" NOT NULL,
    message_date timestamp(6) with time zone DEFAULT now(),
    user_id integer NOT NULL DEFAULT
nextval('gator_message_user_id_seq'::regclass),
    item_id integer NOT NULL DEFAULT
nextval('gator_message_item_id_seq'::regclass),
    CONSTRAINT gator_message_pkey PRIMARY KEY (message_id),
    CONSTRAINT fk_item FOREIGN KEY (item_id)
        REFERENCES public.item (item_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_user_record FOREIGN KEY (user_id)
        REFERENCES public.user_record (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

4.1.5 dashboard - This table is associated with user details. It will have the following data fields.

```
CREATE TABLE public.dashboard
(
    dashboard_id integer NOT NULL DEFAULT
     nextval('dashboard_dashboard_id_seq'::regclass),
    user_id integer NOT NULL DEFAULT nextval('dashboard_user_id_seq'::regclass),
    message_id integer NOT NULL DEFAULT
     nextval('dashboard_message_id_seq'::regclass),
    item_id integer NOT NULL DEFAULT nextval('dashboard_item_id_seq'::regclass),
    CONSTRAINT dashboard_pkey PRIMARY KEY (dashboard_id),
    CONSTRAINT fk_gator_message FOREIGN KEY (message_id)
       REFERENCES public.gator_message (message_id) MATCH SIMPLE
       ON UPDATE NO ACTION
       ON DELETE NO ACTION,
    CONSTRAINT fk_item FOREIGN KEY (item_id)
       REFERENCES public.item (item_id) MATCH SIMPLE
       ON UPDATE NO ACTION
       ON DELETE NO ACTION,
    CONSTRAINT fk_user_record FOREIGN KEY (user_id)
       REFERENCES public.user_record (user_id) MATCH SIMPLE
       ON UPDATE NO ACTION
       ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;
```

5. List of requested P1 functions – provided by your CEO

The flowing functions constitute minimal P1 list and you must implement and test them for December demo

- Home page
- Search (with search text validation)
- Search results
- Item details
- Log in and registration
- Item posting
- Message from buyer to seller
- Seller dashboard (for seller items and messages)
- Admin dashboard (backup is to use DB native admin tool but must show ability to approve posted images)