



**DALHOUSIE  
UNIVERSITY**

FACULTY OF  
COMPUTER SCIENCE

## **USER SCENARIO**



# **PlantOne**

**CSCI 5308**

**ADVANCED TOPICS IN SOFTWARE DEVELOPMENT**

**WINTER 2022**

**INSTRUCTOR**

**DR. TUSHAR SHARMA**

**TEACHING ASSISTANT**

**MANJINDER SINGH**

### **GROUP 12**

**SARTHAK PANDIT**

B00900388, [sr215260@dal.ca](mailto:sr215260@dal.ca)

**FENIL PARMAR**

B00895684, [fn902491@dal.ca](mailto:fn902491@dal.ca)

**JAY PATEL**

B00906433, [iy977367@dal.ca](mailto:iy977367@dal.ca)

**AMAN PATEL**

B00888136, [am501483@dal.ca](mailto:am501483@dal.ca)

**SAHIL PAREKH**

B00900956, [sh883193@dal.ca](mailto:sh883193@dal.ca)

## **Social Module**

- **User Registration:**
  - **User Sign Up:** The social module is used for creating and viewing user information, and events. The home screen gives an option to sign up or log in in the welcome screen. New users can register using Sign up feature, by entering the following fields as part of sign-up form -
    - i.**Email address:** Used for authentication during log-in. Used as unique id for each user. The email should adhere to standard email address formats. No two users can have the same email address.
    - ii.**Password**
    - iii.**First Name**
    - iv.**Last Name**
  - **User Log In:** Existing user can log in with the registered email address and password the existing users can log in to the portal.
- **User Profile:**
  - **View & Edit Profile:** The users can view their own profile with the field information they entered in sign up form and additionally update/add/ edit the following fields -
    - i.**First Name**
    - ii.**Last Name**
    - iii.**Email address**
    - iv.**Date of Birth**
    - v.**Street**
    - vi.**City**
    - vii.**Country**
    - viii.**Postal Code**
    - ix.**Profile Picture**
- **User Security:**
  - **JWT Token:** The user session is authenticated using JSON Web Token, which is a URL-safe way to transfer message between two parties, ensuring authenticity and integrity.
  - **Forgot password:** In an event if user forgets the password, a reset password link is sent on their registered email address. This is to ensure that all the sensitive user information is communicated on the optimum medium.
  - **Password encoder:** The password stored in the backend database is encrypted using Spring security Default Password Encoder module.
- **Events:** Once the user is successfully able to log in, the user can view **Events** using the global navigation panel.
  - **Create & View Events:** Under the Events section, the user can create new events and view all the events posted by other users.

Each event will have the following fields –

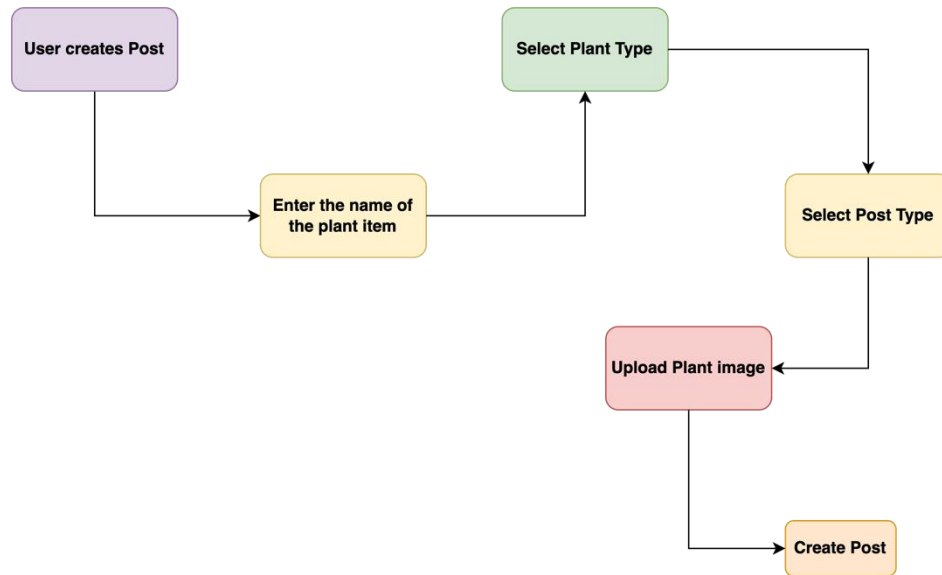
- i. **Event Title**
- ii. **Event Detail text**
- iii. **Event mode (in-person/ online)**
- iv. **Event Link**
- v. **Maximum seats**
- vi. **Available seats**

- **Event Details:** Each event has a detail view, to view all the above fields along with booking/registering for an event.
- **Event Registration:** Once the user has registered to an event, a unique registration id is allotted as part of each booking. As a successful confirmation for the registration, a ticket is generated with the **event registration details** in a downloadable PDF.
- **Delete Events:** On the event listing page, the users can delete the events they created. Only the permissible users can perform this action.

## **Exchange Module –**

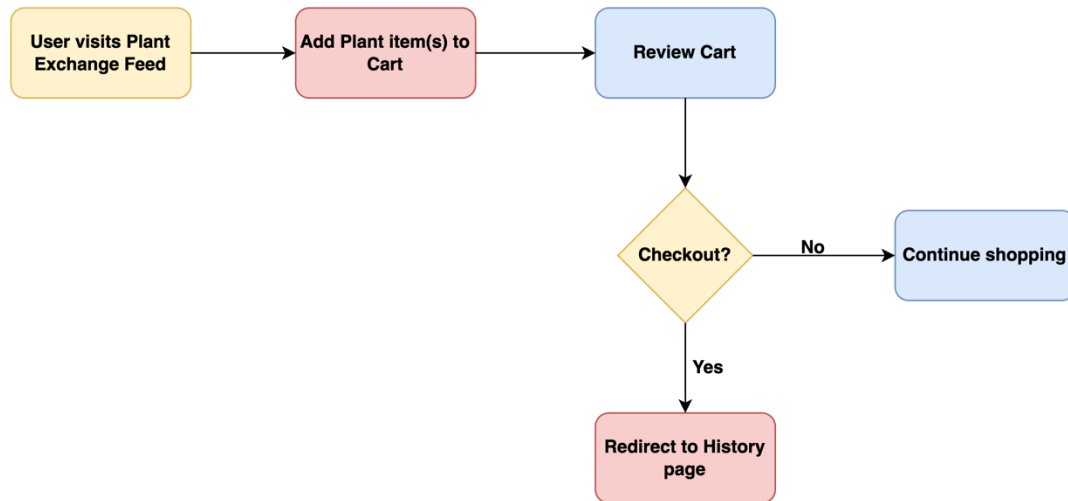
- **Create Posts:** Users can create posts for selling their plants related products. Users will be given options to customize their products into categories including type of plants and type of posts into the following categories -

- **Plant Type –**
  - i. **Succulent**
  - ii. **Temperate**
  - iii. **Tropical and subtropical**
  - iv. **Forced bulbs**
- **Post Types –**
  - i. **Buy**
  - ii. **Adopt**
  - iii. **Exchange**
- **Plant Image -**
- Using the above categories, users can perform the following operations for their ease of use in viewing posts -
  - i. **Filter posts**
  - ii. **Sort posts**
  - iii. **Search posts**



*Flowchart 1 - User scenario – Create Post*

- **Like Posts:** Users can hit likes in the posts put for sale by other users. Users can also like their own posts.
- **Revert like:** Users can unlike the posts and revert their previous like selection.
- **Comments:** Users can comment on the posts and interact with the sellers and other users to negotiate the price and availability of the product being put out for sale.
- **Delete comments:** Users can delete their previous comments on the posts.
- **Add posts to cart:** Users can add their preferred products to cart. The cart can hold multiple items that can be checked out together.
- **Delete individual cart entries:** Users can delete individual cart entries from their cart.
- **Checkout cart entries:** Users can check out their cart entries. Upon checkout, all the cart entries will be considered sold and will not be available to other users and will not be visible on the exchange feed.
- **Maintaining consistent state of database:** No two users can check out the same post. If the same post is present in the cart for more than one user, the post will be checked out on first come-first-serve basis. Once the post is checked out by any user, it will automatically be deleted from the cart for all the other users. This feature ensures a consistent state of database and avoids dirty read in transactions.



*Flowchart 2 - User Scenario – Checkout cart*

## **Knowledge Module –**

- **Create blogs:** Users can create blog and express their thoughts and ideas and get a huge visibility from the audience.
- **View blogs:** Users can view blogs written by other users. The users will also be able to read their submitted version of blog in this view.
- **Hit upvote on blogs:** Users can upvote on the blogs written by other users.
- **Revert upvote:** Users can revert their upvotes on the blogs.

# BLOG

Feature	Add blog
Request	Blog name, Article, userId
Response	Blog object
Api reference	POST <a href="/plantone/api/v1/blog/">/plantone/api/v1/blog/</a>
Description	Users can create blog and express their thoughts and ideas and get a huge visibility from the audience.

```
public Blog addBlog(Blog blog)
{
    String result;
    try{
        blog.setVoteCount(0);
        blog.setCreatedDate(Instant.now());
        blogRepository.save(blog);
    }catch (Exception exception)
    {
        throw new GlobalException("Exception: "+exception, false);
    }
    return blog;
}
```

Figure 1: Code snipped for addBlog

PlantOne
Events
Knowledge
Exchange
Profile
Sign out

# Create a Blog

Title:

Blog Description:

B
I
U
S

"
"
<
>
H1
H2

|
|
|
|

x
x
x
x

Normal

Normal

A

Sans Serif

I
x

Type your post...

Create

Figure 2: Screenshot for Create a blog

Feature	Delete blog
Request	Blog UUID
Response	Blog object
Api reference	DELETE <a href="/plantone/api/v1/blog/{blogUUID}">/plantone/api/v1/blog/{blogUUID}</a>
Description	Users can delete the blog.

```

public Blog deleteBlogById(UUID blogUUID) throws EmptyResultDataAccessException {
    Blog blog = new Blog();
    try{
        blog = blogRepository.getById(blogUUID);
        blogRepository.deleteById(blogUUID);
    }catch(Exception exception){
        throw new GlobalException("Exception: "+exception, false);
    }
    return blog;
}

```

Figure 3: Code snipped for deleteBlogById method

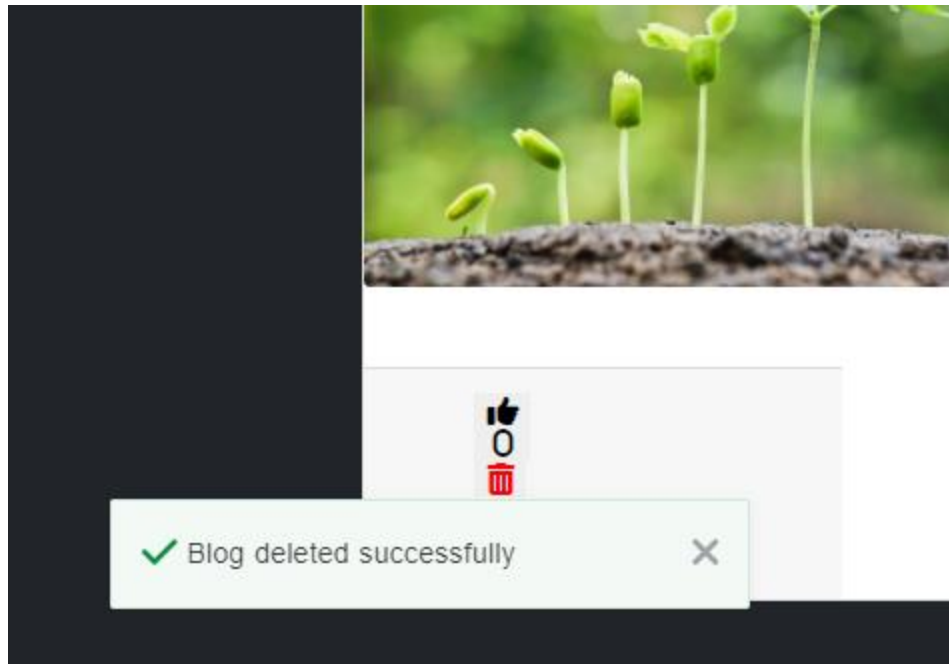


Figure 4: Screen Shot for delete the blog

Feature	Get all blogs
Request	User Id
Response	List of Blog response objects
Api reference	GET <a href="#">/plantone/api/v1/blog/getAllBlogs/{user_id}</a>
Description	Users can view blogs written by other users. The users will also be able to read their submitted version of blog in this view.



```

public List<BlogResponse> getBlogsByUser(UUID user_id) {
    List<Blog> blogList = blogRepository.findAll();
    List<BlogResponse> blogResponsesList = new ArrayList<>();
    for (Blog blog:blogList) {
        BlogResponse blogResponse = new BlogResponse();
        blogResponse = CustomModelMapper.blogResponseModelMapper(blog);

        Integer isLiked = voteRepository.isUserAlreadyUpvoted(blog.getBlogUUID(),user_id);
        if(isLiked == 1)
        {
            blogResponse.setLikedByUser(true);
        }else
        {
            blogResponse.setLikedByUser(false);
        }
        blogResponsesList.add(blogResponse);
    }
    return blogResponsesList;
}

```

Figure 5: Code snippet for getBlogsByUser

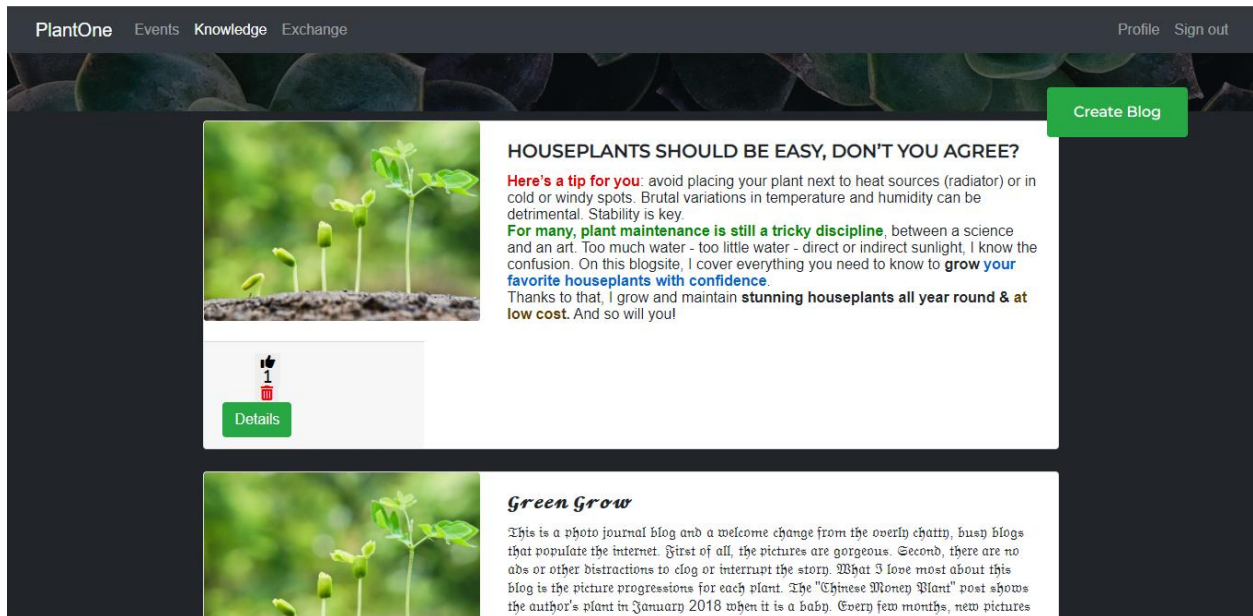


Figure 6: Screenshot for getting all the blogs

Feature	Get blog
Request	Blog UUID
Response	Blog object

Api reference	GET <a href="/plantone/api/v1/blog/{blogUUID}">/plantone/api/v1/blog/{blogUUID}</a>
Description	Users can view blog written by other users by specific id.

```
public Blog getBlog(UUID blogUUID)
{
    Optional<Blog> blog = blogRepository.findById(blogUUID);
    return blog.get();
}
```

Figure 7: Code snipped for getBlog

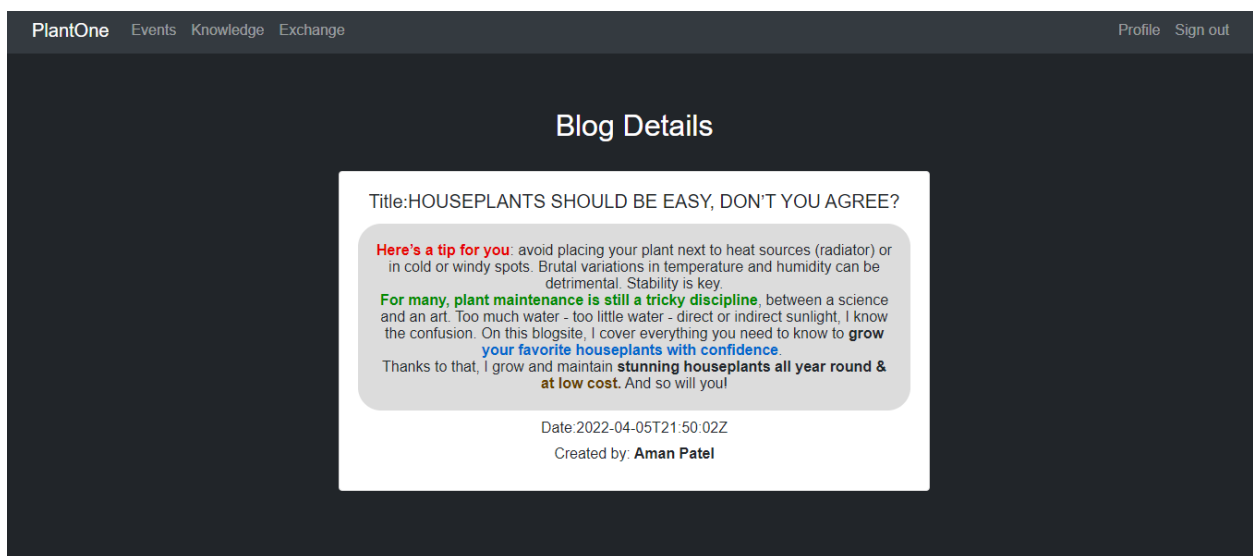


Figure 8: Screenshot of Detail page of blog

## CART ENTRY

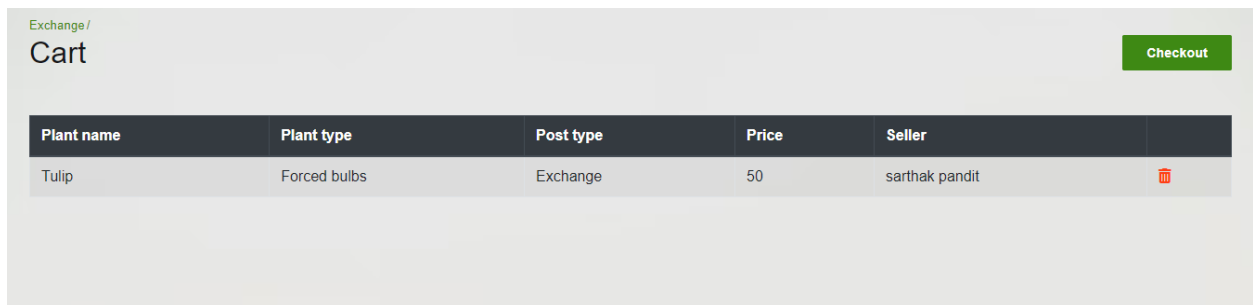
**Maintaining a consistent state of the database:** No two users can check out the same post. If the same post is present in the cart for more than one user, the post will be checked out on a first-come-first-serve basis. Once the post is checked out by any user, it will automatically be deleted from the cart for all the other users. This feature ensures a consistent state of the database and avoids dirty reads in transactions.

Feature	Get Cart Items
---------	----------------

Request	User Id
Response	List of cart entry
Api reference	GET <a href="/plantone/api/v1/cart/getCartItems/{user_id}">/plantone/api/v1/cart/getCartItems/{user_id}</a>
Description	User can get the list of all the items in their cart.

```
public List<CartEntry> getCartItems(UUID userUUID){
    return cartEntryRepository.getCartItems(userUUID);
}
```

Figure 9: Code snipped for getCartItems method



Exchange / Cart						Checkout
Plant name	Plant type	Post type	Price	Seller		
Tulip	Forced bulbs	Exchange	50	sarthak pandit		

Figure 10: Screenshot for cart items

Feature	Add to cart entry
Request	Post Id and User Id
Response	Cart entry object
Api reference	POST <a href="/plantone/api/v1/cart/add/{user_id}/{post_id}">/plantone/api/v1/cart/add/{user_id}/{post_id}</a>
Description	User can add post of sellers to their cart.

```

public CartEntry addToCartEntry(UUID postUUID, UUID userUUID)
{
    CartEntry cartEntry = new CartEntry();
    cartEntry.setCart(cartRepository.findByUser(userUUID));
    cartEntry.setPost(postRepository.findBypostUUID(postUUID));
    //cartEntry.setUser(userRepository.getById(userUUID));
    cartEntry.setCheckedOut(false);
    cartEntry.setSoftDelete(false);
    return cartEntryRepository.save(cartEntry);
}

```

Figure 11: Code snipped for addToCartEntry

Feature	Delete Cart Entry
Request	Cart Id
Response	String
Api reference	DELETE <a href="/plantone/api/v1/cart/deleteCartItem/{cart_entry_id}">/plantone/api/v1/cart/deleteCartItem/{cart_entry_id}</a>
Description	User can delete individual cart entried from their cart.

```

public String deleteCartEntry(UUID cartUUID){
    String result;
    try{
        cartEntryRepository.deleteById(cartUUID);
        result = "Cart entry deleted successfully";
    }catch(Exception exception){
        result = "Exception:"+exception;
    }
    return result;
}

```

Figure 12: Code snipped for deleteCartEntry

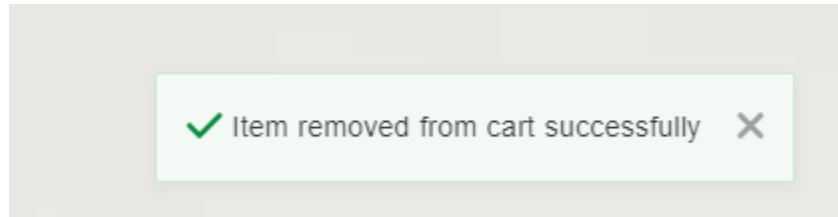


Figure 13: Screenshot for removing the item from cart

Feature	Checkout cart entry
Request	User Id
Response	List of cart entry
Api reference	POST <a href="#">/plantone/api/v1/cart/checkout/{user_id}</a>
Description	Users can check out their cart entries. Upon checkout, all the cart entries will be considered sold and will not be available to other users and will not be visible on the exchange feed.

```

public List<CartEntry> checkoutCartEntry(UUID userUUID) {

    Cart cart = cartRepository.findByUser(userUUID);
    UUID cartUUID = cart.getCartUUID();
    List<UUID> postUUID = cartEntryRepository.findByCart(cartUUID);

    int numberOfPosts = postUUID.size();
    int itemsAvailable = 0;
    for (UUID singlePost : postUUID) {
        UUID expr = cartEntryRepository.isCheckedOut(singlePost);
        if (expr == null) {
            itemsAvailable++;
        }
    }

    if (itemsAvailable == numberOfPosts) //All the items in the cart are available
    {
        for (UUID checkOutPost : postUUID) { // Now check out

            cartEntryRepository.updateCheckOutStatus(cartUUID, checkOutPost);
            postRepository.updatePostBuyer(userUUID, checkOutPost); //The post is now bought

        }
    } else if (itemsAvailable != numberOfPosts) {

        for (UUID checkOutPost : postUUID) {

            cartEntryRepository.softDeleteItems(checkOutPost, cartUUID);
        }
    }

    return cartEntryRepository.findByCartUUID(cartUUID);
}

```

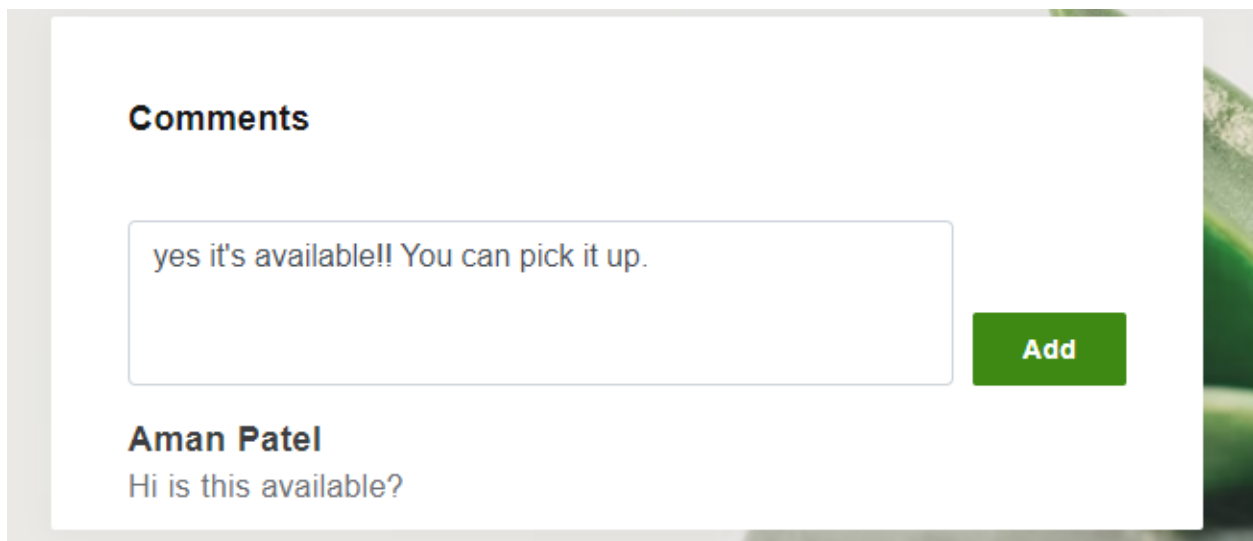
*Figure 14: Code snippet for checkoutCartEntry*

## Comment

Feature	Add comment
Request	Post Id and Comment object
Response	Comment Object
Api reference	POST <a href="#">/plantone/api/v1/comment/add/{post_id}</a>
Description	Users can comment on the posts and interact with the sellers and other users to negotiate the price and availability of the product being put out for sale.

```
public Comment addComment(UUID postUUID, Comment comment){
    Optional<User> user = userRepository.findById(comment.getUser().getUser_id());
    comment.setUser(user.get());
    comment.setPost(postRepository.findById(postUUID));
    return commentRepository.save(comment);
}
```

Figure 15: Code snipped for addComment



**Comments**

yes it's available!! You can pick it up.

**Add**

**Aman Patel**  
Hi is this available?

Figure 16: Screenshot to add the comment

Feature	Delete Comment
Request	Comment Id

Response	Comment will be deleted
Api reference	DELETE <a href="/plantone/api/v1/comment/{comment_id}">/plantone/api/v1/comment/{comment_id}</a>
Description	Users can delete their previous comments on the posts.

```
public void deleteComment(UUID commentUUID){
    commentRepository.deleteByCommentUUID(commentUUID);
}
```

Figure 17: Code snippet for deleteComment

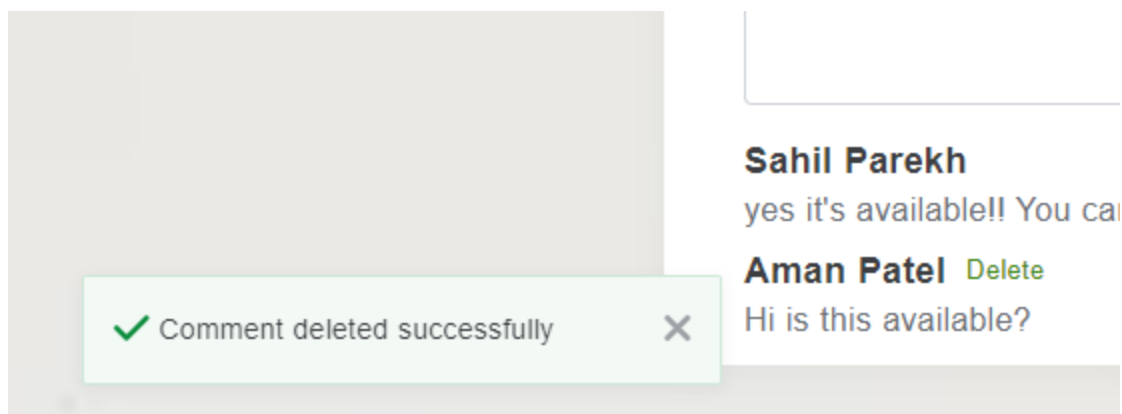


Figure 18: Screenshot to delete the comment

## Event

Feature	Get all events
Request	Get events
Response	List of events
Api reference	GET <a href="/plantone/api/v1/event/">/plantone/api/v1/event/</a>
Description	On the event listing page, the users can delete the events they created. Only the permissible users can perform this action.



```
public List<Event> getEvents(){
    return eventRepository.findAll();
}
```

Figure 19: Code snipped for getEvents

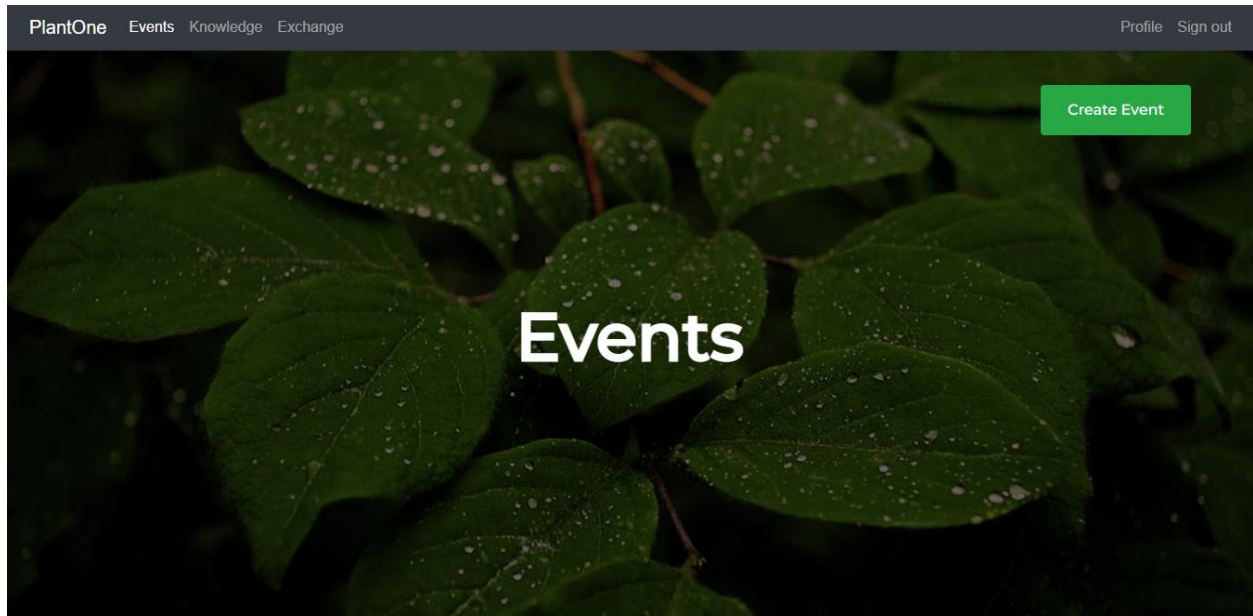


Figure 20: Screenshot for event page

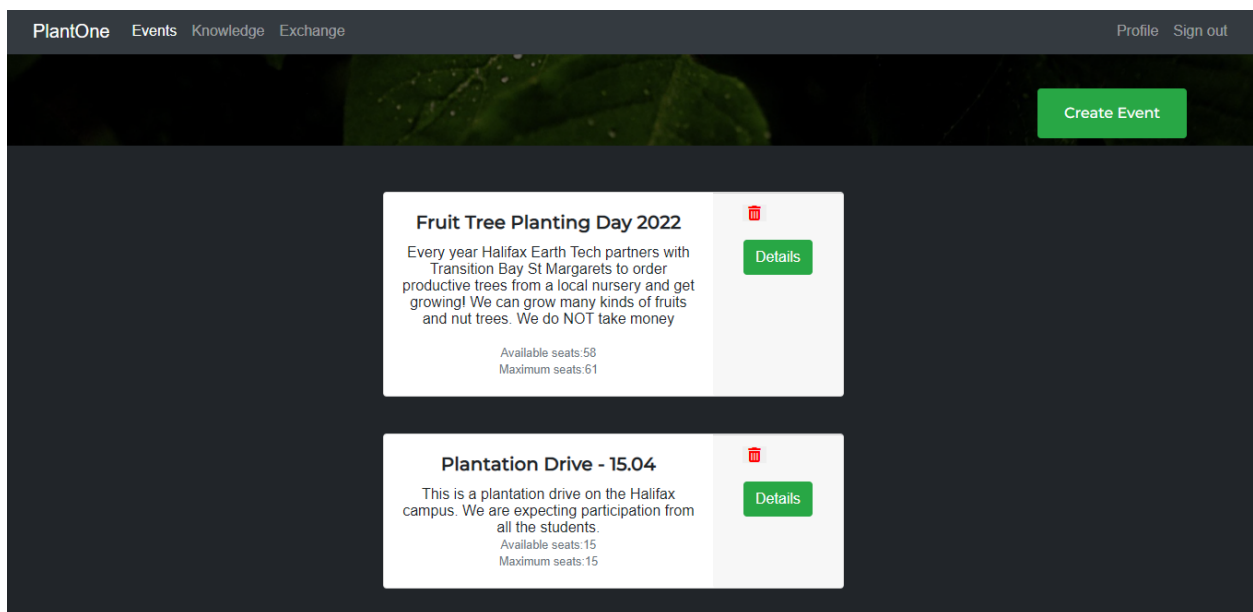


Figure 21: Screenshot for getting all the events

Feature	Add event
Request	Event object
Response	Event object
Api reference	POST <a href="/plantone/api/v1/event/add">/plantone/api/v1/event/add</a>
Description	<p>Under the Events section, the user can create new events and view all the events posted by other users.</p> <p>Each event will have the following fields –</p> <ul style="list-style-type: none"> <li>• <b>Event Title</b></li> <li>• <b>Event Detail text</b></li> <li>• <b>Event mode (in-person/ online)</b></li> <li>• <b>Event Link</b></li> <li>• <b>Maximum seats</b></li> <li>• <b>Available seats</b></li> </ul>

```
public Event addEvent(Event event)
{
    event.setAvailable_seats(event.getMaximum_seats());
    return eventRepository.save(event);
}
```

Figure 22: Code snippet for addEvent

PlantOne Events Knowledge Exchange Profile Sign out

## Create an Event

Title :

Event date:

Link :

Maximum Seats :

Mode :

Event Description:

Sans Serif

Type your post...

Figure 23: Screenshot for creating the event

Feature	Get event
Request	Event id
Response	Event
Api reference	GET <a href="/plantone/api/v1/event/{event_id}">/plantone/api/v1/event/{event_id}</a>
Description	Each event has a detail view, to view all the above fields along with booking/registering for an event.

```
public Event getEvent(UUID eventUUID)
{
    Optional<Event> event = eventRepository.findById(eventUUID);
    return event.get();
}
```

Figure 24: Code snippet for `getEvent`

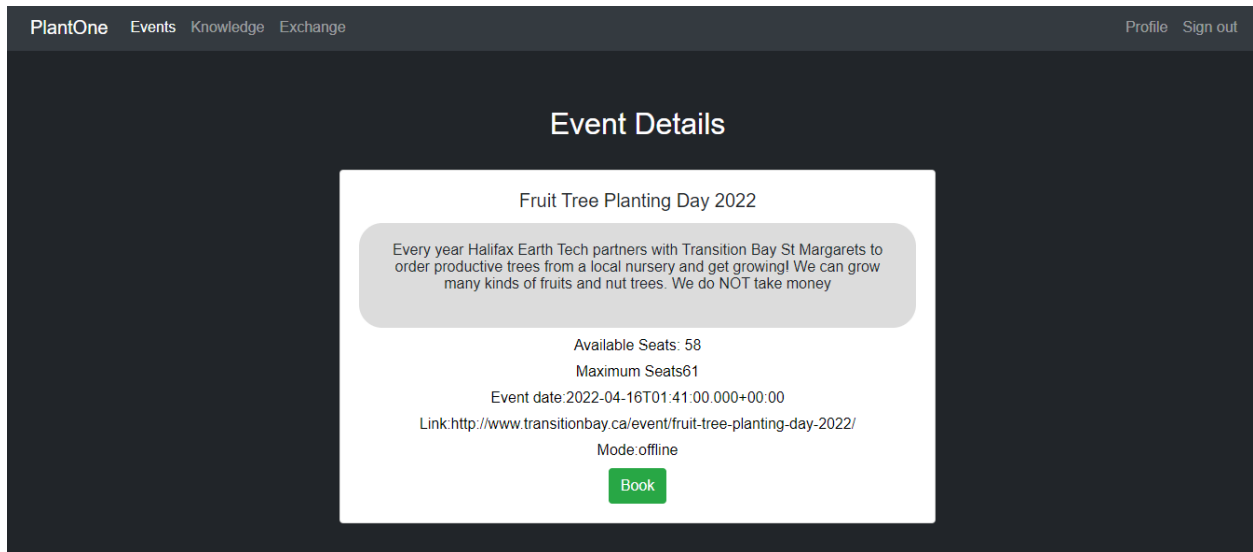


Figure 25: Screenshot for event details page

Feature	Register Event
Request	Event Registration object
Response	Event will be successfully registered
Api reference	POST <a href="/plantone/api/v1/event/register/{event_id}">/plantone/api/v1/event/register/{event_id}</a>

Description	Once the user has registered to an event, a unique registration id is allotted as part of each booking. As a successful confirmation for the registration, a ticket is generated with the <b>event registration details</b> in a downloadable PDF.
-------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

public String registerEvent(EventRegistrations eventRegistrations){

    UUID eventUUID = eventRegistrations.getEvent().getEventUUID();
    if(checkAvailableSeats(eventUUID)) {
        String usernameRegistrationId = eventRegistrations.getUser().getUsername();
        String uuidRegistrationId = eventUUID.toString().substring(EVENT_UUID_BEGIN_INDEX,EVENT_UUID_END_INDEX)
            + randomUUID().toString().substring(EVENT_UUID_BEGIN_INDEX,RANDOM_EVENT_UUID_END_INDEX);
        String registrationId = usernameRegistrationId + uuidRegistrationId;
        eventRegistrations.setRegistrationId(registrationId);
        eventRegistrationRepository.save(eventRegistrations);
        return "Your registration is successful.\nPlease note your registration id is: " + registrationId;
    }
    else
        return "Sorry, all seats have been booked.";
}

```

Figure 26: Code snippet for registerEvent

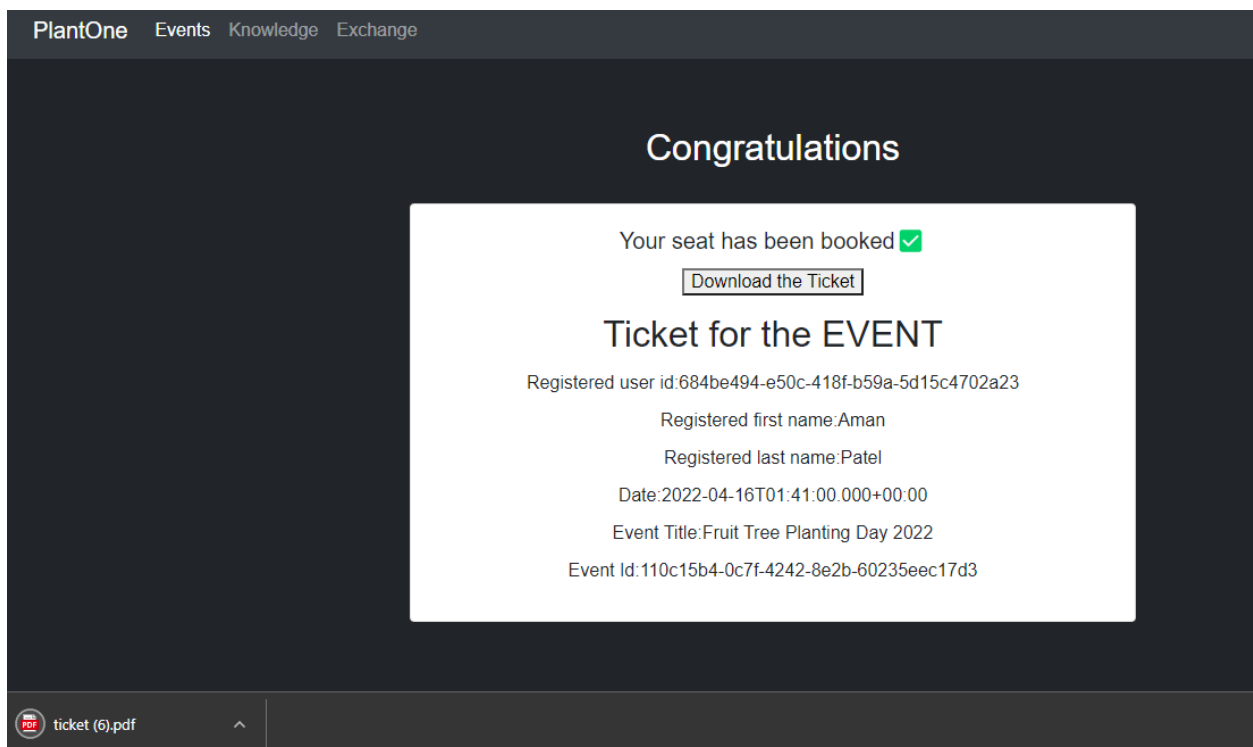


Figure 27: Screenshot for registering the event

Feature	Check available Seats
Request	Event Id
Response	Boolean
Description	If the event has available seats user can register the event.

```

public boolean checkAvailableSeats(UUID eventUUID){

    Optional <Event> event = eventRepository.findById(eventUUID);
    int availableSeats = event.get().getAvailable_seats();
    if(availableSeats > MIN_AVAILABLE_SEATS){
        availableSeats--;
        event.get().setAvailable_seats(availableSeats);
        return true;
    }
    else
        return false;
}

```

Figure 28: Code snippet for checkAvailableSeats

Feature	Delete event
Request	Event id
Response	Event will be successfully deleted
Api reference	DELETE <a href="/plantone/api/v1/event/{event_id}">/plantone/api/v1/event/{event_id}</a>
Description	On the event listing page, the users can delete the events they created. Only the permissible users can perform this action.

```

public String deleteEventById(UUID event_id) throws EmptyResultDataAccessException {
    String result;
    try{
        eventRepository.deleteById(event_id);
        result = "Event deleted successfully";
    }catch(Exception exception){
        result = "Exception:"+exception;
    }
    return result;
}

```

Figure 29: Code snippet for deleteEventById

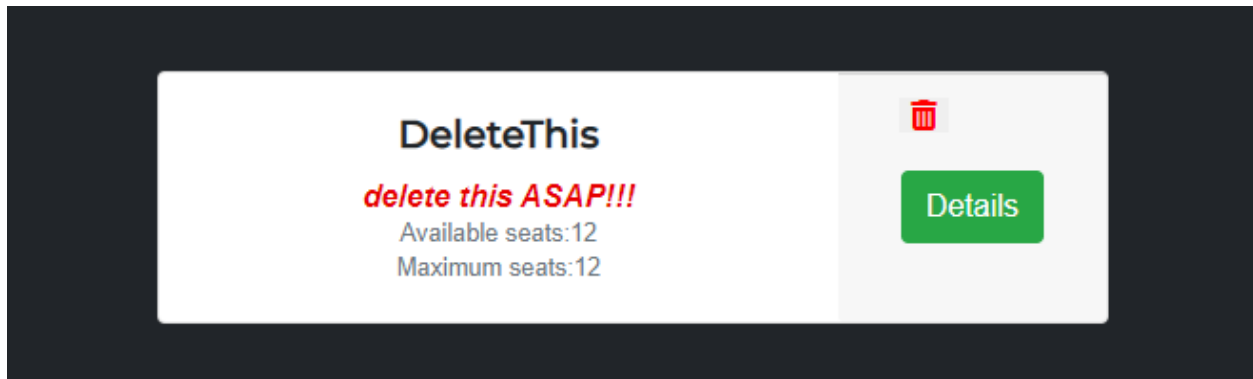


Figure 30: Screenshot for deleting the event

## Likes

Feature	Like Click
Request	Post id, User id
Response	Like will be registered on post
Api reference	POST <a href="#">/plantone/api/v1/post/like/{postUUID}/{user_id}</a>
Description	Users can hit likes in the posts put for sale by other users. Users can also like their own posts.

```

@Transactional
public String likesClick(UUID postUUID, UUID userUUID) {

    Post post = postRepository.getById(postUUID);
    User user = userRepository.getById(userUUID);
    Likes likes = new Likes(post,user);
    String result;

    if(likesRepository.isUserAlreadyLiked(postUUID,userUUID)>0)
    {
        likesRepository.deleteById(likesRepository.getLikeIdByUserAndPost(postUUID, userUUID));
        post.setLikesCount(post.getLikesCount() - 1);
        result = "Like removed successfully";
    }else
    {
        likesRepository.save(likes);
        post.setLikesCount(post.getLikesCount() + 1);
        result = "Liked successfully";
    }

    postRepository.save(post);
    return result;
}

```

Figure 31: Code snippet for likesClick

## Password

Feature	Create Password reset token for User
Request	User object, Token
Response	Token will be generated
Api reference	POST <a href="#">/plantone/api/v1/passwordReset/changePassword</a>
Description	User can generate the password reset token by raising the request on forgot password.

```

@Override
public void createPasswordResetTokenForUser(final User user, final String token) {
    final PasswordResetToken myToken = new PasswordResetToken(token, user);
    passwordTokenRepository.save(myToken);
}

```

Figure 32: Code snippet for createPasswordResetTokenForUser

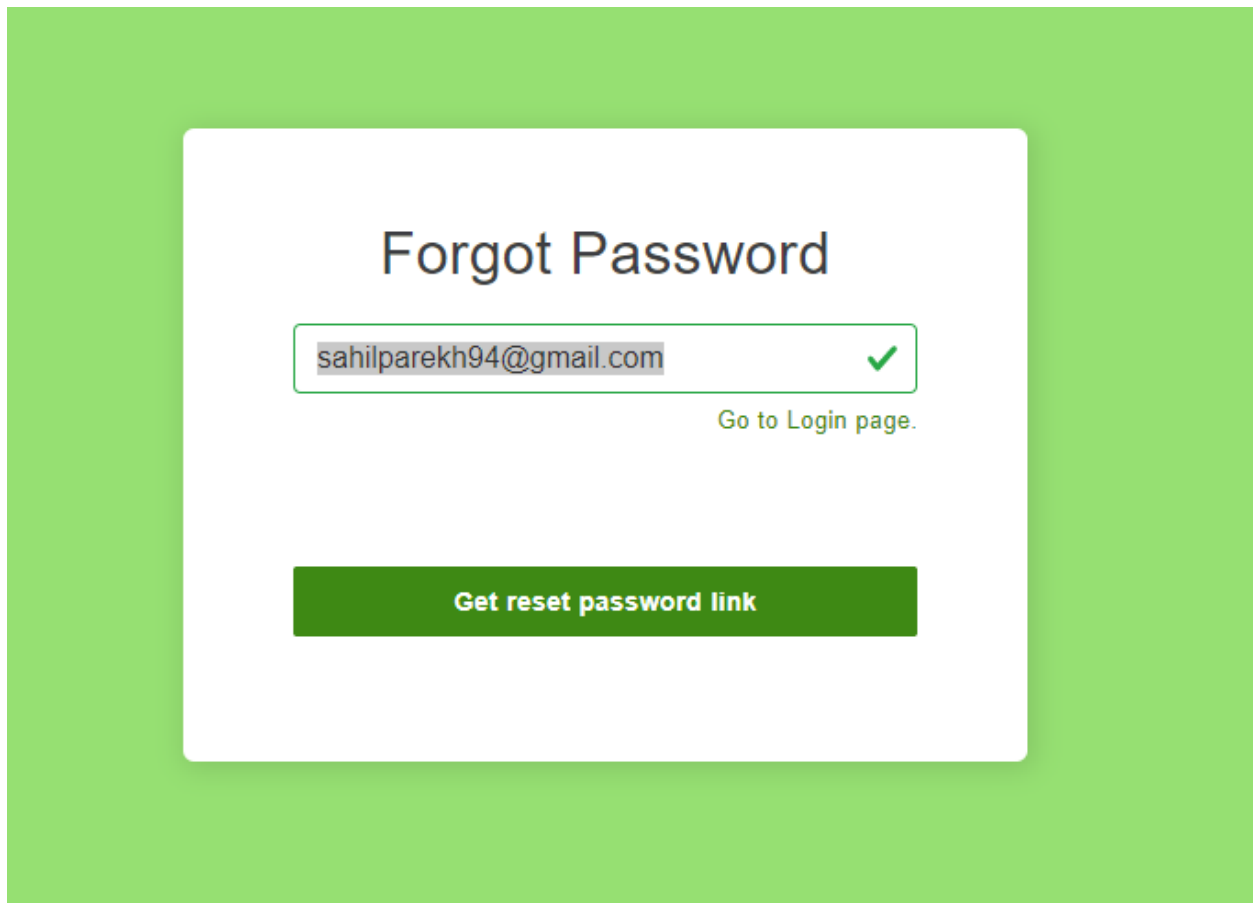


Figure 33: Screenshot for createPasswordResetTokenForUser

Feature	Get password reset Token
Request	Token
Response	Password Reset Token
Description	User will get the password reset token on their mail. The user session is authenticated using JSON Web Token, which is a URL-safe way to transfer message between two parties, ensuring authenticity and integrity.

```
@Override
public PasswordResetToken getPasswordResetToken(final String token) {
    return passwordTokenRepository.findByToken(token);
}
```

Figure 34: Code snippet for getPasswordResetToken



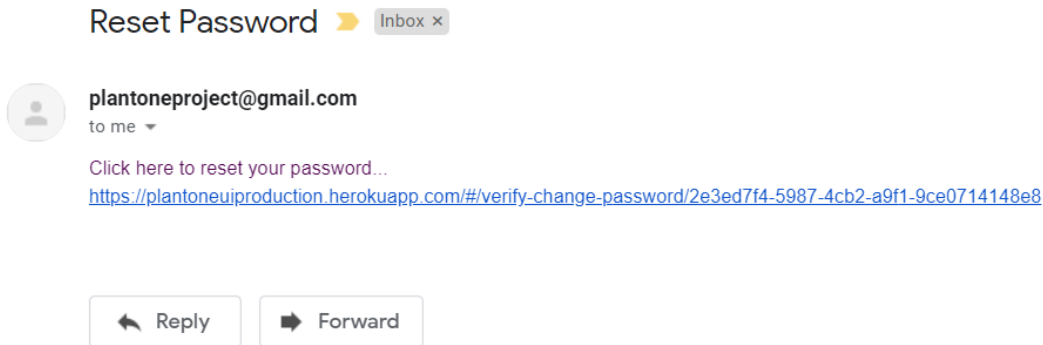


Figure 35: Screenshot for getPasswordResetToken

Feature	Get user by password reset token
Request	Token
Response	User object
Description	This methods checks if user is present by the reset token generated.

```
@Override
public Optional<User> getUserByPasswordResetToken(final String token) {
    return Optional.ofNullable(passwordTokenRepository.findByToken(token).getUser());
}
```

Figure 36: Code snippet for getUserByPasswordResetToken method

Feature	Change User Password
Request	User, Password dto object
Response	User object
Api reference	POST <a href="#">/plantone/api/v1/passwordReset/resetPassword</a>
Description	In an event if user forgets the password, a reset password link is sent on their registered email address. This is to ensure that all the sensitive user information is communicated on the optimum medium.

```
@Override
public void changeUserPassword(final User user, PasswordDto passwordDto) {
    user.setPassword(passwordEncoder.encode(passwordDto.getNewPassword()));
    userRepository.save(user);
    passwordTokenRepository.deleteByToken(passwordDto.getToken());
}
```

Figure 37: Code snippet for checkUserPassword

Change Password

\*\*\*\* ✓

\*\*\*\* ✓

[Go to Login page.](#)

**Save**

© 2022 Plant One. All Rights Reserved.

Figure 38: Screenshot for Change password

Feature	Delete password reset token
Request	Date
Response	Password token will be deleted
Description	After the expiration time the reset token will be deleted.

```
@Override
public void deletePasswordResetToken(Date now){
    passwordTokenRepository.deleteByExpiryDateLessThan(now);
}
```

Figure 39: Code snippet for deletePasswordResetToken

Feature	Validate password reset token
Request	Token
Response	Get status of validation
Api reference	GET <a href="#">/plantone/api/v1/passwordReset/verifyToken</a>
Description	This method validate the password reset token.

```
@Override
public String validatePasswordResetToken(String token) {
    final PasswordResetToken passToken = passwordTokenRepository.findByToken(token);
    String status = !isTokenFound(passToken) ? TOKEN_INVALID
        : isTokenExpired(passToken) ? TOKEN_EXPIRED
        : "";
    return getMessage(status);
}
```

Figure 40: Code snippet for validatePasswordResetToken

Feature	Is token found
Request	Password reset token object
Response	Boolean
Description	This method checks if the token used to reset the password is present or not.

```
private boolean isTokenFound(PasswordResetToken passToken) {
    return passToken != null;
}
```

Figure 41: Code snippet for validatePasswordResetToken

Feature	Is token expired
Request	Password reset token object
Response	Boolean
Description	This method check whether the token is expired or not.

```
private boolean isTokenExpired(PasswordResetToken passToken) {
    final Calendar cal = Calendar.getInstance();
    return passToken.getExpiryDate().before(cal.getTime());
}
```

Figure 42: Code snippet for isTokenExpired

Feature	Get Message
Request	String of result
Response	Message will be returned in string form
Description	This functionality return the status of the message.

```
private String getMessage(String result){
    String message = "";
    if(result.equalsIgnoreCase(TOKEN_EXPIRED)){
        message = "Your registration token has expired. Please register again.";
    }else if(result.equalsIgnoreCase(TOKEN_INVALID)){
        message = "Invalid token.";
    }
    return message;
}
```

Figure 43: Code snippet for getMessage

## Post

Feature	Get post list
Request	User id
Response	List of post response
Api reference	GET <a href="/plantone/api/v1/post/allPosts/{user_id}">/plantone/api/v1/post/allPosts/{user_id}</a>
Description	This method return the list of post responses created by sellers.

```

public List<PostResponse> getPostList(UUID userUUID) {

    //Get only active post from all the other sellers
    List<Post> postList = postRepository.getActivePosts(userUUID);
    List<PostResponse> postResponsesList = new ArrayList<>();
    for (Post post:postList) {
        PostResponse postResponse = new PostResponse();

        postResponse = CustomModelMapper.postResponseModelMapper(post);
        if(post.getPlantImage()!=null){
            String encodeToString = "data:image/"+post.getFileExtension()+";base64," + Base64.getEncoder().
            postResponse.setBase64Image(encodeToString);
        }

        Integer isLiked = likesRepository.isUserAlreadyLiked(post.getPostUUID(),userUUID);
        if(isLiked == 1)
        {
            postResponse.setLikedByUser(true);
        }else
        {
            postResponse.setLikedByUser(false);
        }
        postResponsesList.add(postResponse);
    }
    return postResponsesList;
}

```

Figure 44: Code snippet for getPostList

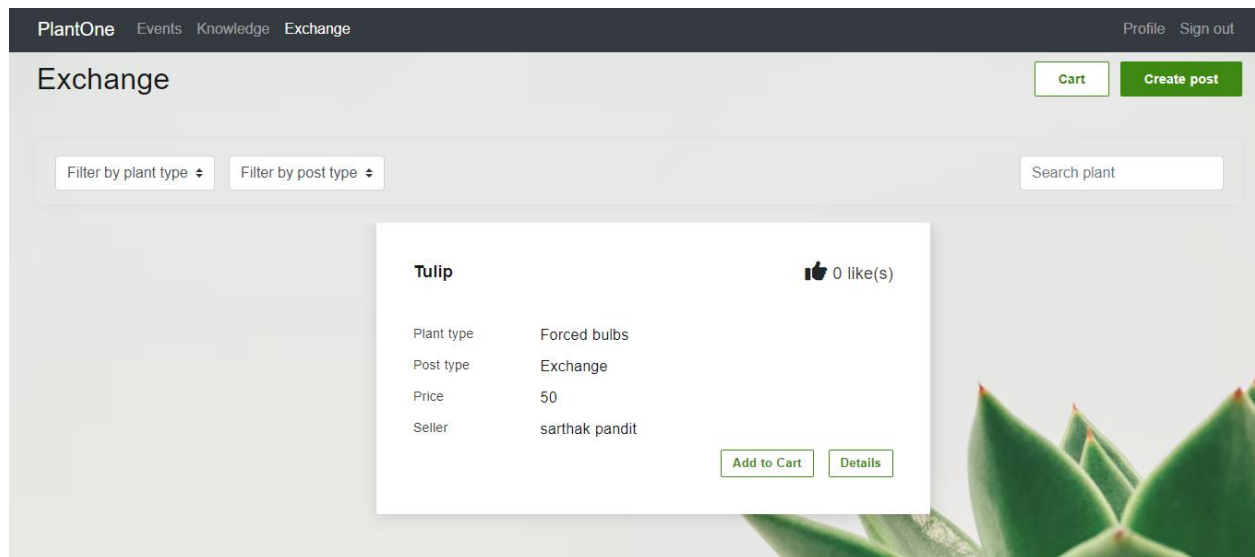


Figure 45: Screenshot of post page

Feature	Add post
Request	Post object, Photo file

Response	Post object
Api reference	POST <a href="#">/plantone/api/v1/post/add</a>
Description	<p>Users can create posts for selling their plants related products. Users will be given options to customize their products into categories including type of plants and type of posts into the following categories -</p> <ul style="list-style-type: none"> <li>○ <u>Plant Type –</u> <ul style="list-style-type: none"> <li>i. Succulent</li> <li>ii. Temperate</li> <li>iii. Tropical and subtropical</li> <li>iv. Forced bulbs</li> </ul> </li> <li>○ <u>Post Types –</u> <ul style="list-style-type: none"> <li>i. Buy</li> <li>ii. Adopt</li> <li>iii. Exchange</li> </ul> </li> <li>○ <u>Plant Image -</u></li> <li>○ Using the above categories, users can perform the following operations for their ease of use in viewing posts - <ul style="list-style-type: none"> <li>i. Filter posts</li> <li>ii. Sort posts</li> <li>iii. Search posts</li> </ul> </li> </ul>

```

public Post addPost(Post post, MultipartFile file ) throws IOException {
    post.setLikesCount(0);
    post.setTimestamp(Date.from(Instant.now()));

    if(file!=null){
        post.setFileExtension(FilenameUtils.getExtension(file.getOriginalFilename()));
        post.setPlantImage(file.getBytes());
    }

    return postRepository.save(post);
}

```

Figure 46: Code snippet for addPost

**Create post**

[Add image](#)

Plant name

Plant type

Post type

[Cancel](#) [Save](#)

Figure 47: Screenshot for creating the post

Feature	Get post
Request	Post id
Response	Post object
Api reference	GET <a href="/plantone/api/v1/post/{post_id}">/plantone/api/v1/post/{post_id}</a>
Description	This method return the post object when details of the post is required by the buyer / seller.

```

public Post getPost(UUID postUUID) {
    Optional<Post> post = postRepository.findById(postUUID);
    Post responsePost = post.get();
    if(responsePost.getPlantImage()!=null){
        String encodeToString = "data:image/"+responsePost.getFileExtension()+";base64," + Base64.getEncoder().
        responsePost.setBase64Image(encodeToString);
    }
    return responsePost;
}

```

Figure 48: Code snippet for getPost

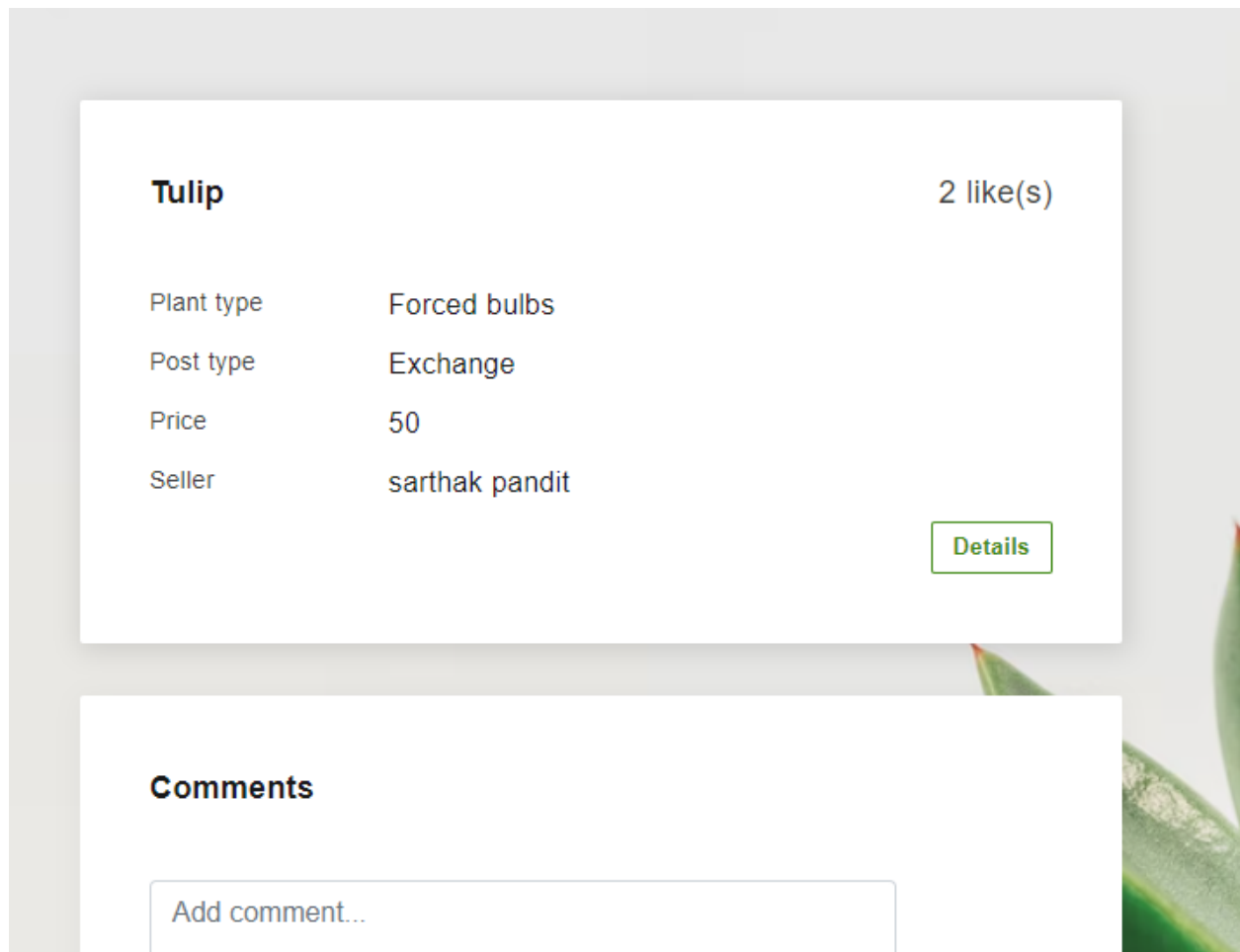


Figure 49: Screenshot for detail page for post

Feature	Delete post
Request	Post id
Response	Status message will be returned
Api reference	DELETE <a href="/plantone/api/v1/post/{post_id}">/plantone/api/v1/post/{post_id}</a>
Description	Seller can delete the created post.



```

public String deletePostById(UUID postUUID) throws EmptyResultDataAccessException {
    String result;
    try{
        postRepository.deleteById(postUUID);
        result = "Post deleted successfully";
    }catch(Exception exception){
        result = "Exception:"+exception;
    }
    return result;
}

```

Figure 50: Code snippet for deletePostById

## User

Feature	Add user
Request	User request, Photo file
Response	User response
Api reference	POST <a href="/plantone/api/v1/users/">/plantone/api/v1/users/</a>
Description	<p>The social module is used for creating and viewing user information, and events. The home screen gives an option to sign up or log in in the welcome screen. New users can register using Sign up feature, by entering the following fields as part of sign-up form -</p> <ol style="list-style-type: none"> <li>i. <b>Email address:</b> Used for authentication during log-in. Used as unique id for each user. The email should adhere to standard email address formats. No two users can have the same email address.</li> <li>ii. <b>Password</b></li> <li>iii. <b>First Name</b></li> <li>iv. <b>Last Name</b></li> </ol>

```

public UserResponse addUser(UserRequest userRequest, MultipartFile file) throws IOException {
    userRequest.setPassword(passwordEncoder.encode(userRequest.getPassword()));
    User user = CustomModelMapper.createRequestModelMapper(userRequest,file);
    User returnUser = userRepository.save(user);
    UserResponse userResponse = CustomModelMapper.responseModelMapper(returnUser);
    return userResponse;
}

```

Figure 51: Code snippet for addUser

Plant One is platform to make more plants available to people at a cheaper price that will encourage more people to contribute towards the good cause.

This platform is an opportunity for plant enthusiasts to build an online community for exchanging ideas and plants.

## Register to Plant One

amxnpatel@gmail.com

....

Aman

Patel

**Sign up**

Already have an account? [Login](#)

© 2021 Plant One. All Rights Reserved.

Figure 52: Screenshot for SignUp

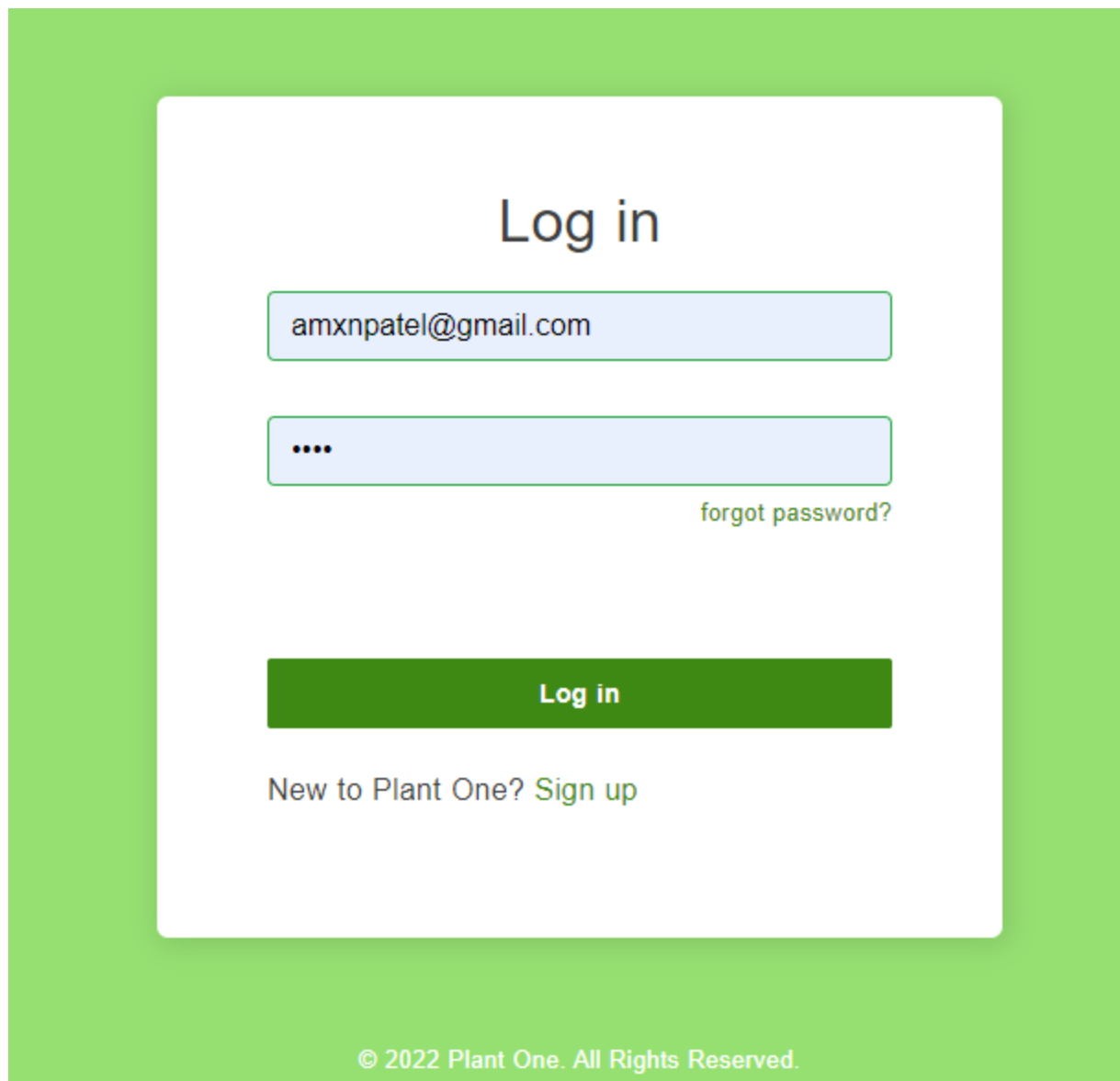


Figure 53: Screenshot for Login

Feature	Update user
Request	User update request and Photo file
Response	User Response
Api reference	PUT <a href="#">/plantone/api/v1/users/</a>
Description	<ul style="list-style-type: none"> <li>The users can view their own profile with the field information they entered in sign up form and additionally update/add/ edit the following fields -</li> </ul>

	<ul style="list-style-type: none"> <li>i. <b>First Name</b></li> <li>ii. <b>Last Name</b></li> <li>iii. <b>Email address</b></li> <li>iv. <b>Date of Birth</b></li> <li>v. <b>Street</b></li> <li>vi. <b>City</b></li> <li>vii. <b>Country</b></li> <li>viii. <b>Postal Code</b></li> <li>ix. <b>Profile Picture</b></li> </ul>
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

public UserResponse updateUser(UserUpdateRequest userUpdateRequest, MultipartFile file) throws IOException {
    User originalUser;
    UserResponse userResponse;

    if (userRepository.findById(userUpdateRequest.getUser_id()).isPresent()) {
        originalUser = userRepository.findById(userUpdateRequest.getUser_id()).get();
        originalUser = CustomModelMapper.updateRequestModelMapper(userUpdateRequest, originalUser, file);
        originalUser = userRepository.save(originalUser);
        userResponse = CustomModelMapper.responseModelMapper(originalUser);
    } else {
        throw new GlobalException("User does not exists.", false);
    }
    return userResponse;
}

```

Figure 54: CodeSnipped for UpdateUser

The screenshot shows a web form for updating a user profile. The form is titled 'AP' and has a 'Change' link. It contains input fields for First name (Aman), Last name (Patel), Email (amxnpatel@gmail.com), Date of birth (yyyy-mm-dd), Street (Enter street), City (Enter city), Country (Enter country), and Postal code (Enter postal code). There are 'Cancel' and 'Save' buttons at the bottom right.

Figure 55: Screen shot for UpdateUser

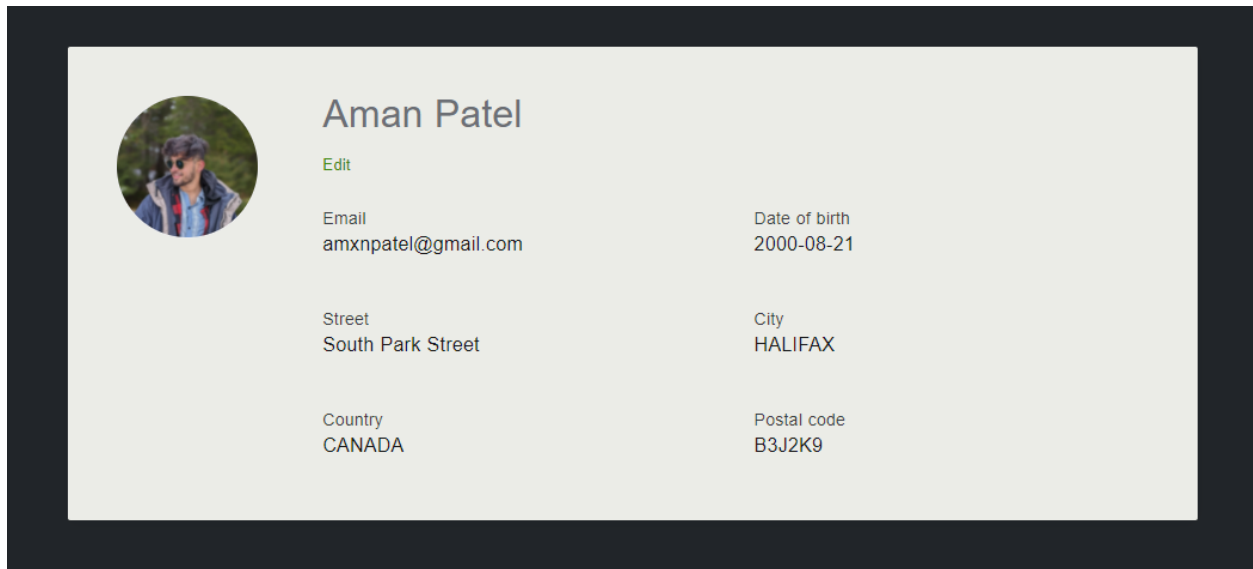


Figure 56: Screenshot for UpdateUser

Feature	Get user by Id
Request	User id
Response	User Response
Api reference	POST <a href="#">/plantone/api/v1/users/{user_id}</a>
Description	This method returns the user by user_id.

```

public UserResponse getUserById(UUID user_id) {
    User user;
    UserResponse userResponse;
    if (userRepository.findById(user_id).isPresent()) {
        user = userRepository.findById(user_id).get();
        userResponse = CustomModelMapper.responseModelMapper(user);
    } else {
        throw new GlobalException("User id does not exists.",false);
    }
    return userResponse;
}

```

Figure 57: CodeSnipped for UserResponse

Feature	Delete user by id
Request	User id

Response	String response
Api reference	DELETE <a href="/plantone/api/v1/users/{user_id}">/plantone/api/v1/users/{user_id}</a>
Description	This method delete the user by their user_id.

```
public String deleteUserById(UUID user_id) throws EmptyResultDataAccessException{
    String result;
    try{
        userRepository.deleteById(user_id);
        result = "User deleted successfully";
    }catch(Exception exception){
        result = "Exception:"+exception;
    }
    return result;
};
```

Figure 58: CodeSnipped for deleteUserById

Feature	Get user by email
Request	Email
Response	User response
Api reference	GET <a href="/plantone/api/v1/users/email/{email}">/plantone/api/v1/users/email/{email}</a>
Description	This method returns the user object by their email id.

```
@Transactional(propagation = Propagation.MANDATORY)
public UserResponse getUserByEmail(String email){
    User user = userRepository.findByEmail(email);
    UserResponse userResponse;
    if (user != null) {
        userResponse = CustomModelMapper.responseModelMapper(user);
        return userResponse;
    } else {
        throw new UsernameNotFoundException("User not found with email: " + email);
    }
}
```

Figure 59: CodeSnipped for getUserByEmail

# Upvote

Feature	Upvote click
Request	Blog id and User id
Response	Upvote status in string format
Api reference	POST <a href="#">/plantone/api/v1/blog/upvote/{blogUUID}/{user_id}</a>
Description	Users can upvote on the blogs written by other users.

```
@Transactional
public String upVoteClick(UUID blogUUID, UUID userUUID) {

    Blog blog = blogRepository.getById(blogUUID);
    User user = userRepository.getById(userUUID);
    Vote vote = new Vote(blog,user);
    String result;

    if(voteRepository.isUserAlreadyUpvoted(blogUUID,userUUID)>0)
    {
        voteRepository.deleteById(voteRepository.getVoteIdByUserAndBlog(blogUUID, userUUID));
        blog.setVoteCount(blog.getVoteCount() - 1);
        result = "Upvote removed successfully";
    }else
    {
        voteRepository.save(vote);
        blog.setVoteCount(blog.getVoteCount() + 1);
        result = "Upvoted successfully";
    }

    blogRepository.save(blog);
    return result;
}
```

Figure 60: CodeSnipped for upVoteClick