Speculative Generality

Identified Code Snippet:

1. `UserAppDao.java`:

```
/**
 * User app DAO.
 *
 * @author jtremeaux
 */
public class UserAppDao {
    /**
     * Create a new connection.
     *
     * @param userApp User app
     * @return ID
     */
    public String create(UserApp userApp) {
        // Implementation
        return null; // Placeholder return
    }

    /**
     * Delete a connection.
     *
     * @param id User app ID
     */
    public void delete(String id) {
        // Implementation
    }

    /**
     * Deletes connection linked to a user and an application.
     *
     * @param userId User ID
     * @param appId App ID
     */
    public void deleteByUserIdAndAppId(String userId, String appId) {
        // Implementation
    }

    /**
     * Search and returns a non-deleted connection.
     *
     * @param id User app ID
     * @return User app
     */
    public UserApp getActiveById(String id) {
        // Implementation
```

```
        return null; // Placeholder return
    }

    /**
     * Search and returns a non-deleted connection by user and app.
     *
     * @param userId User ID
     * @param appId App ID
     * @return User app
     */
    public UserApp getActiveByUserIdAndAppId(String userId, String
appId) {
        // Implementation
        return null; // Placeholder return
    }

    /**
     * Search and returns application list with connection status.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findByUserId(String userId) {
        // Implementation
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns user's connected applications.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findConnectedByUserId(String userId) {
        // Implementation
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns applications connected to a user.
     *
     * @param appId Application ID
     * @return User app list
     */
    public List<UserAppDto> findByAppId(String appId) {
        // Implementation
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Updates a connection.
     *
     * @param userApp User app
     * @return Updated user app
```

```
     */
    public UserApp update(UserApp userApp) {
        // Implementation
        return null; // Placeholder return
    }
}
```

2. `UserContactDao.java`:

```
/**
 * User's contact DAO.
 *
 * @author jtremeaux
 */
public class UserContactDao {
    /**
     * Create a new contact.
     *
     * @param userContact UserContact
     * @return ID
     */
    public String create(UserContact userContact) {
        //implementation
    }

    /**
     * Returns users' contact.
     *
     * @param userId User ID
     * @param appId Application ID
     * @return Users' contact
     */
    @SuppressWarnings("unchecked")
    public List<UserContactDto> findByUserIdAndAppId(String userId,
String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select uc.USC_ID_C,
uc.USC_EXTERNALID_C ");
        sb.append(" from T_USER_CONTACT uc");
        sb.append(" where uc.USC_IDUSER_C = :userId and uc.USC_IDAPP_C
= :appId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userId", userId);
        q.setParameter("appId", appId);
        List<Object[]> l = q.getResultList();
        List<UserContactDto> userContactDtoList = new
ArrayList<UserContactDto>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
```

```
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDtoList.add(userContactDto);
        }

        return userContactDtoList;
    }

    /**
     * Updates last check user contact date.
     *
     * @param userId User ID
     * @param appId Application ID
     */
    public void updateByUserIdAndAppId(String userId, String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("update T_USER_CONTACT
uc");
        sb.append(" set uc.USC_UPDATEDATE_D = :updateDate ");
        sb.append(" where uc.USC_IDUSER_C = :userId and uc.USC_IDAPP_C
= :appId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("updateDate", new Date());
        q.setParameter("userId", userId);
        q.setParameter("appId", appId);
        q.executeUpdate();
    }

    /**
     * Delete a contact.
     *
     * @param id User contact ID
     */
    public void delete(String id) {
        //implementation
    }

    /**
     * Search user's contacts.
     *
     * @param criteria Search criteria
     * @param paginatedList Paginated list (filled by side effect)
     */
    public void findByCriteria(PaginatedList<UserContactDto>
paginatedList, UserContactCriteria criteria) {
        Map<String, Object> parameterMap = new HashMap<String, Object>
();

        StringBuilder sb = new StringBuilder("select uc.USC_ID_C,
uc.USC_EXTERNALID_C, uc.USC_FULLNAME_C ");
        sb.append(" from T_USER_CONTACT uc");
```

```
        // Add search criterias
        List<String> criteriaList = new ArrayList<String>();
        criteriaList.add("uc.USC_DELETEDATE_D is null");
        if (criteria.getAppId() != null) {
            criteriaList.add("uc.USC_IDAPP_C = :appId");
            parameterMap.put("appId", criteria.getAppId());
        }
        if (criteria.getUserId() != null) {
            criteriaList.add("uc.USC_IDUSER_C = :userId");
            parameterMap.put("userId", criteria.getUserId());
        }
        if (!Strings.isNullOrEmpty(criteria.getQuery())) {
            criteriaList.add("lower(uc.USC_FULLNAME_C) like
:fullName");
            parameterMap.put("fullName", "%" +
criteria.getQuery().toLowerCase() + "%");
        }

        if (!criteriaList.isEmpty()) {
            sb.append(" where ");
            sb.append(Joiner.on(" and ").join(criteriaList));
        }

        sb.append(" order by uc.USC_FULLNAME_C, uc.USC_ID_C");

        // Perform the search
        QueryParam queryParam = new QueryParam(sb.toString(),
parameterMap);
        List<Object[]> l =
PaginatedLists.executePaginatedQuery(paginatedList, queryParam);

        // Build results
        List<UserContactDto> userContactDtoList = new
ArrayList<UserContactDto>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDto.setFullName((String) o[i++]);
            userContactDtoList.add(userContactDto);
        }
        paginatedList.setResultList(userContactDtoList);
    }
}
```

3. `UserAppCreatedAsyncListener.java`:

```
/**
 * User app created listener.
 *
```

```
 * @author jtremeaux
 */
public class UserAppCreatedAsyncListener {
    /**
     * Logger.
     */
    private static final Logger log =
LoggerFactory.getLogger(UserAppCreatedAsyncListener.class);

    /**
     * Process event.
     *
     * @param userAppCreatedEvent Event
     */
    @Subscribe
    public void onUserCreated(final UserAppCreatedEvent
userAppCreatedEvent) {
        if (log.isInfoEnabled()) {
            log.info("UserApp created event: " +
userAppCreatedEvent.toString());
        }

        final UserApp userApp = userAppCreatedEvent.getUserApp();

        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                try {
                    AppId appId = AppId.valueOf(userApp.getAppId());
                    switch (appId) {
                    case FACEBOOK:
                        // Synchronize friends contact
                        final FacebookService facebookService =
AppContext.getInstance().getFacebookService();
                        String facebookAccessToken =
userApp.getAccessToken();

facebookService.synchronizeContact(facebookAccessToken,
userApp.getUserId());
                        break;
                    }
                } catch (Exception e) {
                    if (log.isErrorEnabled()) {
                        log.error("Error synchronizing App contacts",
userAppCreatedEvent, e);
                    }
                }
            }
        });
    }
}
```

4. `ConnectResource.java`:

```
@Path("/connect")
public class ConnectResource extends BaseResource {

    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        //implementation
    }

    @POST
    @Path("{id: [a-z]+}/add")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(
            @PathParam("id") String appIdString,
            @FormParam("access_token") String accessToken) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Get application to add
        AppId appId = getAppId(appIdString);

        switch (appId) {
        case FACEBOOK:
            // Specific application logic
            //implementation
            break;
        }

        // Always return OK
        //implementation
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}/remove")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(@PathParam("id") String appIdString) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Get application to remove
        AppId appId = getAppId(appIdString);

        // Delete user app for this application
        //implementation

        // Always return OK
        //implementation
```

```
    }

    @POST
    @Path("{id: [a-z0-9\\-]+}/update")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            @PathParam("id") String appIdString,
            @FormParam("sharing") boolean sharing) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Get application to update
        AppId appId = getAppId(appIdString);

        // Check if the user is connected to this application
        //implementation

        // Always return OK
        //implementation
    }

    @GET
    @Path("{id: [a-z0-9\\-]+}/contact/list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response contactList(
            @PathParam("id") String appIdString,
            @QueryParam("query") String query,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) throws JSONException
{
        if (!authenticate()) {
            //implementation
        }

        // Get application
        AppId appId = getAppId(appIdString);

        // Check if the user is connected to the application
        //implementation

        // Search contacts
        //implementation

        return Response.ok().entity(response).build();
    }

    private AppId getAppId(String appIdString) throws JSONException {
        AppId appId = null;
        try {
            appId = AppId.valueOf(StringUtils.upperCase(appIdString));
        } catch (Exception e) {
            throw new ClientException("AppNotFound", "Application not
```

```
found: " + appIdString);
        }
        return appId;
    }
}
```

Apply Manual Refactoring

Refactored code:

1. `UserAppDao.java`:

```
/**
 * User app DAO.
 *
 * @author jtremeaux
 */
public class UserAppDao {
    /**
     * Create a new connection.
     *
     * @param userApp User app
     * @return ID
     */
    public String create(UserApp userApp) {
        // Implementation remains same
        return userApp.getId();
    }

    /**
     * Delete a connection.
     *
     * @param id User app ID
     */
    public void delete(String id) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();

        // Get the user app
        Query q = em.createQuery("select ua from UserApp ua where
ua.id = :id and ua.deleteDate is null");
        q.setParameter("id", id);
        UserApp userAppFromDb = (UserApp) q.getSingleResult();

        // Delete the user app
        userAppFromDb.setDeleteDate(new Date());
    }

    /**
     * Deletes connection linked to a user and an application.
     *
     * @param userId User ID
     */
    public void deleteByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("update UserApp ua set ua.deleteDate
= :deleteDate where ua.userId = :userId and ua.deleteDate is null");
        q.setParameter("deleteDate", new Date());
```

```
            q.setParameter("userId", userId);
            q.executeUpdate();
        }

        /**
         * Search and returns a non-deleted connection.
         *
         * @param id User app ID
         * @return User app
         */
        public UserApp getActiveById(String id) {
            // Implementation remains same
            try {
                // Process Query
                return (UserApp) q.getSingleResult();
            } catch (NoResultException e) {
                return null;
            }
        }

        /**
         * Search and returns a non-deleted connection by user
         *
         * @param userId User ID
         * @return User app
         */
        public UserApp getActiveByUserId(String userId) {
            EntityManager em =
    ThreadLocalContext.get().getEntityManager();
            StringBuilder sb = new StringBuilder("select distinct ua from
    UserApp ua");
            sb.append(" where ua.userId = :userId ");
            sb.append(" and ua.deleteDate is null ");
            sb.append(" order by ua.createDate desc ");
            Query q = em.createQuery(sb.toString());
            try {
                q.setParameter("userId", userId);
                return (UserApp) q.getSingleResult();
            } catch (NoResultException e) {
                return null;
            }
        }

        /**
         * Search and returns application list with connection status.
         *
         * @param userId User ID
         * @return User app list
         */
        public List<UserAppDto> findByUserId(String userId) {
            List<UserAppDto> userAppDtoList = new ArrayList<UserAppDto>();
            // Implementation remains same
            return userAppDtoList;
        }
```

```
    /**
     * Search and returns user's connected applications.
     *
     * @param userId User ID
     * @return User app list
     */
    @SuppressWarnings("unchecked")
    public List<UserAppDto> findConnectedByUserId(String userId) {
        List<UserAppDto> userAppDtoList = new ArrayList<UserAppDto>();
        // Implementation remains same
        return userAppDtoList;
    }

    /**
     * Updates a connection.
     *
     * @param userApp User app
     * @return Updated user app
     */
    public UserApp update(UserApp userApp) {
        //Implementation remains same
        return userApp;
    }
}
```

2. `UserContactDao.java`:

```
/**
 * User's contact DAO.
 *
 * This class focuses on managing user contacts independently of any
specific application context.
 *
 * @author jtremeaux
 */
public class UserContactDao {
    /**
     * Create a new contact.
     *
     * @param userContact UserContact
     * @return ID
     */
    public String create(UserContact userContact) {
        //Implementation remains same
        return userContact.getId()

    }

    /**
     * Returns users' contact.
```

```
     *
     * @param userId User ID
     * @return Users' contact
     */
    @SuppressWarnings("unchecked")
    public List<UserContactDto> findByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select uc.USC_ID_C,
uc.USC_EXTERNALID_C ");
        sb.append(" from T_USER_CONTACT uc");
        sb.append(" where uc.USC_IDUSER_C = :userId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userId", userId);
        List<Object[]> l = q.getResultList();
        List<UserContactDto> userContactDtoList = new
ArrayList<UserContactDto>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDtoList.add(userContactDto);
        }

        return userContactDtoList;
    }

    /**
     * Updates last check user contact date.
     *
     * @param userId User ID
     */
    public void updateByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("update T_USER_CONTACT
uc");
        sb.append(" set uc.USC_UPDATEDATE_D = :updateDate ");
        sb.append(" where uc.USC_IDUSER_C = :userId ");
        sb.append(" and uc.USC_DELETEDATE_D is null ");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("updateDate", new Date());
        q.setParameter("userId", userId);
        q.executeUpdate();
    }

    /**
     * Delete a contact.
     *
     * @param id User contact ID
     */
    public void delete(String id) {
```

```
        // Implementation remains same
    }

    /**
     * Search user's contacts based on criteria.
     *
     * @param criteria Search criteria
     * @param paginatedList Paginated list (filled by side effect)
     */
    public void findByCriteria(PaginatedList<UserContactDto>
paginatedList, UserContactCriteria criteria) {
        // The implementation of this method is specific to the
refactoring effort and addresses the removal of app-specific contexts.
        // Therefore, its detailed implementation is kept as an
example of adapting search functionality to focus on user-centric
criteria.
        Map<String, Object> parameterMap = new HashMap<>();

        StringBuilder sb = new StringBuilder("SELECT uc.USC_ID_C,
uc.USC_EXTERNALID_C, uc.USC_FULLNAME_C ");
        sb.append("FROM T_USER_CONTACT uc ");

        List<String> criteriaList = new ArrayList<>();
        criteriaList.add("uc.USC_DELETEDATE_D IS NULL");
        if (criteria.getUserId() != null) {
            criteriaList.add("uc.USC_IDUSER_C = :userId");
            parameterMap.put("userId", criteria.getUserId());
        }
        if (!Strings.isNullOrEmpty(criteria.getQuery())) {
            criteriaList.add("LOWER(uc.USC_FULLNAME_C) LIKE
:fullName");
            parameterMap.put("fullName", "%" +
criteria.getQuery().toLowerCase() + "%");
        }

        if (!criteriaList.isEmpty()) {
            sb.append("WHERE ");
            sb.append(String.join(" AND ", criteriaList));
        }

        sb.append(" ORDER BY uc.USC_FULLNAME_C, uc.USC_ID_C");

        QueryParam queryParam = new QueryParam(sb.toString(),
parameterMap);
        List<Object[]> l =
PaginatedLists.executePaginatedQuery(paginatedList, queryParam);

        List<UserContactDto> userContactDtoList = new ArrayList<>();
        for (Object[] o : l) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) o[i++]);
            userContactDto.setExternalId((String) o[i++]);
            userContactDto.setFullName((String) o[i]);
```

```
            userContactDtoList.add(userContactDto);
        }
        paginatedList.setResultList(userContactDtoList);
    }
}
```

3. `UserAppCreatedAsyncListener.java`:

```
/**
 * User app created listener.
 *
 * @author jtremeaux
 */
public class UserAppCreatedAsyncListener {
    /**
     * Logger.
     */
    private static final Logger log =
LoggerFactory.getLogger(UserAppCreatedAsyncListener.class);

    /**
     * Process event.
     *
     * @param userAppCreatedEvent Event
     */
    @Subscribe
    public void onUserCreated(final UserAppCreatedEvent
userAppCreatedEvent) {
        if (log.isInfoEnabled()) {
            log.info("UserApp created event: " +
userAppCreatedEvent.toString());
        }

        final UserApp userApp = userAppCreatedEvent.getUserApp();

        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                try {

                    // Synchronize friends contact
                    final FacebookService facebookService =
AppContext.getInstance().getFacebookService();
                    String facebookAccessToken =
userApp.getAccessToken();

facebookService.synchronizeContact(facebookAccessToken,
userApp.getUserId());

                } catch (Exception e) {
```

```
                    if (log.isErrorEnabled()) {
                        log.error("Error synchronizing App contacts",
userAppCreatedEvent, e);
                    }
                }
            }
        });
    }
}
```

4. ConnectResource.java:

```
/**
 * Connected application REST resource.
 *
 * @author jtremeaux
 */
@Path("/connect")
public class ConnectResource extends BaseResource {
    /**
     * Returns current user's connected applications.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Search connected applications
        UserAppDao userAppDao = new UserAppDao();
        List<UserAppDto> userAppList =
userAppDao.findByUserId(principal.getId());

        List<JSONObject> items = new ArrayList<JSONObject>();
        for (UserAppDto userAppDto : userAppList) {
            JSONObject userApp = new JSONObject();
            userApp.put("connected", userAppDto.getId() != null &&
userAppDto.getAccessToken() != null);
            userApp.put("username", userAppDto.getUsername());
            userApp.put("sharing", userAppDto.isSharing());
            items.add(userApp);
        }

        JSONObject response = new JSONObject();
        response.put("apps", items);
        return Response.ok().entity(response).build();
    }
```

```
    /**
     * Add a connected application.
     *
     * @param authToken OAuth authorization token
     * @return Response
     * @throws JSONException
     */
    @POST
    @Path("{id: [a-z]+}/add")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(@FormParam("access_token") String accessToken)
throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Validate input data
        accessToken =
ValidationUtil.validateStringNotBlank(accessToken, "access_token");

        UserAppDao userAppDao = new UserAppDao();
        UserApp userApp = null;
        // Delete old connection to this application
        userAppDao.deleteByUserId(principal.getId());

        // Exchange the short lived token (2h) for a long lived one
(60j)
        final FacebookService facebookService =
AppContext.getInstance().getFacebookService();
        String extendedAccessToken = null;
        try {
            extendedAccessToken =
facebookService.getExtendedAccessToken(accessToken);
        } catch (AuthenticationException e) {
            throw new ClientException("InvalidAuthenticationToken",
"Error validating authentication token", e);
        }

        // Check permissions
        try {
            facebookService.validatePermission(extendedAccessToken);
        } catch (PermissionException e) {
            throw new ClientException("PermissionNotFound",
e.getMessage(), e);
        }

        // Create the connection to the application
        userApp = new UserApp();
        userApp.setAccessToken(extendedAccessToken);
        userApp.setUserId(principal.getId());
        userApp.setSharing(true);

        // Get user's personnal informations
```

```
        facebookService.updateUserData(extendedAccessToken, userApp);

        userAppDao.create(userApp);



        // Raise a user app created event
        UserAppCreatedEvent userAppCreatedEvent = new
UserAppCreatedEvent();
        userAppCreatedEvent.setUserApp(userApp);

AppContext.getInstance().getAsyncEventBus().post(userAppCreatedEvent);

        // Always return OK
        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    /**
     * Remove a connected application.
     *
     * @param appIdString App ID
     * @return Response
     * @throws JSONException
     */
    @POST
    @Path("{id: [a-z0-9\\-]+}/remove")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(
            @PathParam("id") String appIdString) throws JSONException
{
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Delete user app for this application
        UserAppDao userAppDao = new UserAppDao();
        userAppDao.deleteByUserId(principal.getId());

        // Always return OK
        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    /**
     * Updates connected application.
     *
     * @param appIdString App ID
     * @param sharing If true, share on this application
     * @return Response
     * @throws JSONException
     */
```

```
    @POST
    @Path("{id: [a-z0-9\\-]+}/update")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            @PathParam("id") String appIdString,
            @FormParam("sharing") boolean sharing) throws
JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Check if the user is connected to this application
        UserAppDao userAppDao = new UserAppDao();
        UserApp userApp =
userAppDao.getActiveByUserId(principal.getId());
        if (userApp == null) {
            throw new ClientException("AppNotConnected",
java.text.MessageFormat.format("You are not connected to the app {0}",
appIdString));
        }

        // Update the user app
        userApp.setSharing(sharing);
        userAppDao.update(userApp);

        // Always return OK
        JSONObject response = new JSONObject();
        response.put("status", "ok");
        return Response.ok().entity(response).build();
    }

    /**
     * Returns contact list on a connected application.
     *
     * @param appIdString App ID
     * @param limit Page limit
     * @param offset Page offset
     * @return Response
     * @throws JSONException
     */
    @GET
    @Path("{id: [a-z0-9\\-]+}/contact/list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response contactList(
            @PathParam("id") String appIdString,
            @QueryParam("query") String query,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) throws JSONException
{
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        // Check if the user is connected to the application
```

```
        UserAppDao userAppDao = new UserAppDao();
        UserApp userApp =
userAppDao.getActiveByUserId(principal.getId());
        if (userApp == null) {
            throw new ClientException("AppNotConnected",
java.text.MessageFormat.format("You are not connected to the app {0}",
appIdString));
        }

        JSONObject response = new JSONObject();
        List<JSONObject> contacts = new ArrayList<JSONObject>();

        // Search contacts
        PaginatedList<UserContactDto> paginatedList =
PaginatedLists.create(limit, offset);
        UserContactCriteria criteria = new UserContactCriteria();
        criteria.setUserId(principal.getId());
        criteria.setQuery(query);
        UserContactDao userContactDao = new UserContactDao();
        userContactDao.findByCriteria(paginatedList, criteria);
        for (UserContactDto userContactDto :
paginatedList.getResultList()) {
            JSONObject userContact = new JSONObject();
            userContact.put("id", userContactDto.getId());
            userContact.put("external_id",
userContactDto.getExternalId());
            userContact.put("full_name",
userContactDto.getFullName());
            contacts.add(userContact);
        }

        response.put("total", paginatedList.getResultCount());
        response.put("contacts", contacts);
        return Response.ok().entity(response).build();
    }

    /**
     * Get application ID.
     *
     * @param appIdString Application ID (string format)
     * @return App ID
     * @throws JSONException
     */
    private AppId getAppId(String appIdString) throws JSONException {
        AppId appId = null;
        try {
            appId = AppId.valueOf(StringUtils.upperCase(appIdString));
        } catch (Exception e) {
            throw new ClientException("AppNotFound",
MessageFormat.format("Application not found: {0}", appIdString));
        }
        return appId;
    }
}
```

LLM Refactoring

1. `UserAppDao.java`:

```
/**
 * UserAppDao manages operations on the UserApp entities, ensuring no
direct dependency
 * on the App entity itself. Operations involving app IDs use the
identifier stored within
 * the UserApp entity, maintaining a clear separation from the App
entity.
 */
public class UserAppDao {

    /**
     * Create a new UserApp entity.
     *
     * @param userApp User app to be created.
     * @return The ID of the created UserApp entity.
     */
    public String create(UserApp userApp) {
        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        // Perform the create operation
        return userApp.getId();
    }

    /**
     * Deletes a UserApp entity by its ID.
     *
     * @param id ID of the UserApp to delete.
     */
    public void delete(String id) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        UserApp userApp = em.find(UserApp.class, id);
        if (userApp != null) {
            userApp.setDeleteDate(new Date());
            em.merge(userApp);
        }
    }

    /**
     * Deletes UserApp entities linked to a specific user ID and app
ID.
     * Note: This method operates on the relationship between UserApp
and the
     * identifier of an App, not the App entity directly.
     *
     * @param userId ID of the user.
     * @param appId ID of the app linked to the user.
     */
```

```
    public void deleteByUserIdAndAppId(String userId, String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("UPDATE UserApp ua SET ua.deleteDate
= :deleteDate WHERE ua.userId = :userId AND ua.appId = :appId AND
ua.deleteDate IS NULL");
        q.setParameter("deleteDate", new Date());
        q.setParameter("userId", userId);
        q.setParameter("appId", appId);
        q.executeUpdate();
    }

    /**
     * Retrieves an active (non-deleted) UserApp by its ID.
     *
     * @param id ID of the UserApp.
     * @return The found UserApp, or null if not found.
     */
    public UserApp getActiveById(String id) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        try {
            return em.createQuery("SELECT ua FROM UserApp ua WHERE
ua.id = :id AND ua.deleteDate IS NULL", UserApp.class)
                    .setParameter("id", id)
                    .getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Retrieves active (non-deleted) UserApp entities based solely on
the user ID.
     * This approach abstracts away from any direct linkage to
specific applications,
     * focusing instead on the user's engagement within the system.
     *
     * @param userId The unique identifier of the user.
     * @return List of active UserApp entities for the user, or an
empty list if none are found.
     */
    public List<UserApp> getActiveByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        List<UserApp> userApps = em.createQuery("SELECT ua FROM
UserApp ua WHERE ua.userId = :userId AND ua.deleteDate IS NULL ORDER
BY ua.createDate DESC", UserApp.class)
                                   .setParameter("userId", userId)
                                   .getResultList();
        return userApps;
    }

    /**
```

```
     * Search and returns application list with connection status.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findByUserId(String userId) {
        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns user's connected applications.
     *
     * @param userId User ID
     * @return User app list
     */
    public List<UserAppDto> findConnectedByUserId(String userId) {
        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        return new ArrayList<>(); // Placeholder return
    }

    /**
     * Search and returns all active UserApp entities as DTOs.
     * The parameter appId is taken but not used.
     *
     * @return List of UserAppDto without filtering by appId, as appId
no longer exists.
     */
    @SuppressWarnings("unchecked")
    public List<UserAppDto> findAllActiveUserApps(String appId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("SELECT ua.USA_ID_C,
ua.USA_IDUSER_C, ua.USA_ACCESSTOKEN_C, ua.USA_USERNAME_C,
ua.USA_SHARING_B ");
        sb.append("FROM T_USER_APP ua ");
        sb.append("WHERE ua.USA_DELETEDATE_D IS NULL "); // Removed
the appId filter
        sb.append("ORDER BY ua.USA_CREATEDATE_D ");
        Query q = em.createNativeQuery(sb.toString());

        List<Object[]> resultList = q.getResultList();
        List<UserAppDto> userAppDtoList = new ArrayList<>();
        for (Object[] record : resultList) {
            int i = 0;
            UserAppDto userAppDto = new UserAppDto();
            userAppDto.setId((String) record[i++]);
            userAppDto.setUserId((String) record[i++]);
            userAppDto.setAppId((String) record[i++]); // Consider if
this is still relevant
            userAppDto.setAccessToken((String) record[i++]);
            userAppDto.setUsername((String) record[i++]);
```

```
            Boolean sharing = (Boolean) record[i];
            userAppDto.setSharing(sharing != null && sharing);
            userAppDtoList.add(userAppDto);
        }

        return userAppDtoList;
    }

    /**
     * Updates a connection.
     *
     * @param userApp User app
     * @return Updated user app
     */
    public UserApp update(UserApp userApp) {

        /* Implementation remains same as the original because
        this method doesnot interact with App entity */
        // Perform the update operation
        return userApp;
    }


    /* The methods for findByUserId, findConnectedByUserId, and update
    remain conceptually unchanged from the original, as they do not
directly interact
    with the App entity but rather work with identifiers or attributes
within UserApp. */
}
```

2. UserContactDao.java:

```
/**
 * User's contact DAO.
 *
 * This class has been refactored to remove any dependencies on
specific application contexts,
 * focusing purely on user contact management.
 *
 * @author jtremeaux
 */
public class UserContactDao {
    /**
     * Create a new contact.
     *
     * @param userContact UserContact
     * @return ID
     */
    public String create(UserContact userContact) {
        /* Implementation remains same because the method is not
directly connected to App Entity */
```

```
    }

    /**
     * Returns all contacts for a given user.
     *
     * @param userId User ID
     * @return List of UserContactDto for the specified user
     */
    @SuppressWarnings("unchecked")
    public List<UserContactDto> findByUserId(String userId) {
        EntityManager em =
ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("SELECT uc.USC_ID_C,
uc.USC_EXTERNALID_C ");
        sb.append("FROM T_USER_CONTACT uc ");
        sb.append("WHERE uc.USC_IDUSER_C = :userId ");
        sb.append("AND uc.USC_DELETEDATE_D IS NULL");
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userId", userId);
        List<Object[]> result = q.getResultList();
        List<UserContactDto> userContactDtoList = new ArrayList<>();
        for (Object[] record : result) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) record[i++]);
            userContactDto.setExternalId((String) record[i++]);
            userContactDtoList.add(userContactDto);
        }

        return userContactDtoList;
    }

    /**
     * Update last check date for all contacts of a user.
     *
     * @param userId User ID
     */
    public void updateByUserId(String userId) {
        /* Implementation remains same because the method is not
directly connected to App Entity */
    }

    /**
     * Delete a contact by its ID.
     *
     * @param id User contact ID
     */
    public void delete(String id) {
        /* Implementation remains same because the method is not
directly connected to App Entity */
    }

    /**
     * Search user's contacts based on criteria.
```

```
     *
     * Note: The search criteria no longer includes appId to ensure
independence from any app-specific context.
     *
     * @param paginatedList Paginated list (filled by side effect)
     * @param criteria Search criteria
     */
    public void findByCriteria(PaginatedList<UserContactDto>
paginatedList, UserContactCriteria criteria) {
        // The implementation details for findByCriteria are important
for understanding how the method adapts to the removal of appId,
        // so we will keep this method's implementation visible as it
directly reflects the refactoring effort.
        Map<String, Object> parameterMap = new HashMap<>();

        StringBuilder sb = new StringBuilder("SELECT uc.USC_ID_C,
uc.USC_EXTERNALID_C, uc.USC_FULLNAME_C ");
        sb.append("FROM T_USER_CONTACT uc ");

        List<String> criteriaList = new ArrayList<>();
        criteriaList.add("uc.USC_DELETEDATE_D IS NULL");
        if (criteria.getUserId() != null) {
            criteriaList.add("uc.USC_IDUSER_C = :userId");
            parameterMap.put("userId", criteria.getUserId());
        }
        if (!Strings.isNullOrEmpty(criteria.getQuery())) {
            criteriaList.add("LOWER(uc.USC_FULLNAME_C) LIKE
:fullName");
            parameterMap.put("fullName", "%" +
criteria.getQuery().toLowerCase() + "%");
        }

        if (!criteriaList.isEmpty()) {
            sb.append("WHERE ");
            sb.append(String.join(" AND ", criteriaList));
        }

        sb.append(" ORDER BY uc.USC_FULLNAME_C, uc.USC_ID_C");

        Query q = em.createNativeQuery(sb.toString());
        for (Map.Entry<String, Object> entry :
parameterMap.entrySet()) {
            q.setParameter(entry.getKey(), entry.getValue());
        }

        List<Object[]> results = q.getResultList();
        List<UserContactDto> userContactDtoList = new ArrayList<>();
        for (Object[] result : results) {
            int i = 0;
            UserContactDto userContactDto = new UserContactDto();
            userContactDto.setId((String) result[i++]);
            userContactDto.setExternalId((String) result[i++]);
            if (result.length > i) { // Check if fullName is part of
the result set
```

```
                userContactDto.setFullName((String) result[i]);
            }
            userContactDtoList.add(userContactDto);
        }
        paginatedList.setResultList(userContactDtoList);
    }

    // Additional methods if needed for UserContactDao operations,
focusing solely on user-centric data management.
}
```

3. `UserAppCreatedAsyncListener.java`:

```
/**
 * Listener for user app creation events.
 * This version abstracts away from specific app handling, focusing on
generic event processing.
 *
 * @author jtremeaux
 */
public class UserAppCreatedAsyncListener {
    private static final Logger log =
LoggerFactory.getLogger(UserAppCreatedAsyncListener.class);

    /**
     * Process event.
     *
     * @param userAppCreatedEvent Event
     */
    @Subscribe
    public void onUserCreated(final UserAppCreatedEvent
userAppCreatedEvent) {
        if (log.isInfoEnabled()) {
            log.info("UserApp created event: " +
userAppCreatedEvent.toString());
        }

        TransactionUtil.handle(new Runnable() {
            @Override
            public void run() {
                try {
                    // Handle user app creation generically without
depending on specific app logic

handleUserAppCreation(userAppCreatedEvent.getUserApp());
                } catch (Exception e) {
                    if (log.isErrorEnabled()) {
                        log.error("Error handling user app creation",
e);
                    }
                }
            }
```

```
        });
    }

    /**
     * Handles the creation of a user app in a generic manner.
     * This method can be expanded or modified to introduce specific
logic based on app type
     * without directly coupling to any specific app's API or logic
here.
     *
     * @param userApp The user app involved in the creation event.
     */
    private void handleUserAppCreation(UserApp userApp) {
        // Implement generic handling logic here
        // For example, notify other parts of the system, log the
event, update related records, etc.
        log.info("Handling creation of user app: " + userApp.getId());

        // Optionally, delegate to a service that can handle specific
app types internally

AppServiceHandler.getInstance().processUserAppCreated(userApp);
    }
}
```

4. `ConnectResource.java`:

```
@Path("/connect")
public class ConnectResource extends BaseResource {

    @GET
    @Path("list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response list() throws JSONException {
        //implementation
    }

    @POST
    @Path("/add")
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(@FormParam("access_token") String accessToken)
throws JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Generic application addition logic
        // The specific app handling logic is removed.
        // Implement generic add application logic here
        //implementation

        // Always return OK
```

```
        //implementation
    }

    @POST
    @Path("/remove")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(@FormParam("appIdentifier") String
appIdentifier) throws JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Generic application removal logic
        // The specific app handling logic is removed.
        // Implement generic remove application logic here
        //implementation

        // Always return OK
        //implementation
    }

    @POST
    @Path("/update")
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            @FormParam("appIdentifier") String appIdentifier,
            @FormParam("sharing") boolean sharing) throws
JSONException {
        if (!authenticate()) {
            //implementation
        }

        // Generic application update logic
        // The specific app handling logic is removed.
        // Implement generic update application logic here
        //implementation

        // Always return OK
        //implementation
    }

    @GET
    @Path("/contact/list")
    @Produces(MediaType.APPLICATION_JSON)
    public Response contactList(
            @QueryParam("query") String query,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset) throws JSONException
{
        if (!authenticate()) {
            //implementation
        }

        // Generic contact list fetching
```

```
        // The specific app handling logic is removed.
        // Implement generic contact list fetching logic here
        //implementation

        return Response.ok().entity(response).build();
    }

    // Removed the getAppId method as it's no longer needed in this
generic approach
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- Manual refactoring has introduced a method deleteByUserId which simplifies deletion by user ID, avoiding direct references to app IDs.
- It has also streamlined the getActiveByUserId method to focus solely on user IDs, removing app ID considerations.
- The manual version seems to simplify and directly address the operations without considering app IDs, aligning with a focus on user-centric data management.

LLM-generated Refactoring:

- The LLM-refactored code maintains methods that originally referenced app IDs but modifies their implementations to ignore or bypass these references, reflecting an attempt to abstract away from the App entity while still keeping the method signatures and general structure intact.
- It includes a method findAllActiveUserApps that attempts to generalize fetching of user apps without filtering by app ID, even though it retains an unused appId parameter, indicating an incomplete abstraction from the app ID concept.

Evaluation of Strengths and Weaknesses:

Manual Refactoring:

Strengths:

- Simplification and Focus: Concentrates on user-related operations, making the DAO more straightforward and easier to maintain.
- Reduction of Irrelevant Parameters: Eliminates unnecessary parameters (e.g., appId), which could simplify future development and reduce confusion.

Weaknesses:

- Potential Loss of Functionality: By removing the ability to filter by app ID, it might limit the flexibility of the DAO in scenarios where application-specific data retrieval is necessary.
- Lack of Generalization: Does not introduce generalized methods that could accommodate future requirements without reintroducing app-specific logic.

LLM-generated Refactoring:

Strengths:

- Attempt at Generalization: Tries to create more generalized methods that could potentially accommodate a wider range of use cases.
- Preservation of Method Signatures: Keeps method signatures that reference app IDs, which might ease the transition for existing codebases relying on these methods, even if the actual dependency on app IDs is minimized.

Weaknesses:

- Incomplete Abstraction: Retains unused parameters and method signatures that suggest app-specific operations, which could lead to confusion or the impression of incomplete refactoring.
- Complexity in Understanding: The presence of app ID references, even in a minimized role, may complicate understanding for new developers or those unfamiliar with the historical context of the refactoring effort.

Scenarios:

LLMs Excel:

- In situations where a quick refactoring is needed to prototype or migrate away from certain dependencies while maintaining a resemblance of the original structure.
- When there's a need to balance between removing specific dependencies and retaining API compatibility for a transitional period.

Manual Intervention is Preferable:

- For targeted, purpose-driven refactoring efforts that aim to simplify and clarify the codebase with a clear understanding of future requirements and constraints.
- When the refactoring aims not just to remove dependencies but also to optimize and streamline the codebase for maintainability and scalability.
- Both approaches have their merits, depending on the goals and constraints of the refactoring effort. The choice between manual and LLM-generated refactoring should be guided by the specific needs of the project, including considerations of maintainability, flexibility, and the ease of future development.