# CRUD Operations Guide 📚

## Basic CRUD Operations

### 1. CREATE (C) ➕

#### Standard Model Creation

javascript

```javascript
const User = require('./models/User');

// Create a new user
const newUser = await User.create({
  firstName: 'John',
  lastName: 'Doe',
  email: 'john@example.com',
  userType: 'Giver',
  phone: '+1234567890'
});

console.log('Created user:', newUser.userID);
```

#### Alternative: Instance Method

javascript

```javascript
const user = new User({
  firstName: 'Jane',
  lastName: 'Smith',
  email: 'jane@example.com',
  userType: 'Collector'
});

await user.save(); // Calls User.create() internally
```

#### With Error Handling

javascript

```
try {
  const application = await Application.create({
    userID: 'user123',
    applicationType: 'Account_Verification',
    justification: 'I need to verify my account'
  });

  console.log('Application created:', application.applicationID);
} catch (error) {
  console.error('Creation failed:', error.message);
  // Handle validation errors, duplicate entries, etc.
}
```

## 2. READ (R) 📖

### Find by ID (Primary Key)

javascript

```javascript
const User = require('./models/User');

// Find specific user
const user = await User.findById('user123');

if (user) {
  console.log('Found user:', user.firstName, user.lastName);
} else {
  console.log('User not found');
}
```

### Find by Specific Field

javascript

```
// Find user by email
const user = await User.findByEmail('john@example.com');

// Find posts by type
const wastePosts = await Post.findByType('Waste');

// Find applications by status
const pendingApps = await Application.findByStatus('Pending');
```

### Find Multiple with Filters

```javascript
// Find all collectors
const collectors = await User.findByUserType('Collector');

// Find pickups by collector
const myPickups = await Pickup.findByCollectorID('collector123');

// Find comments on a specific post
const postComments = await Comment.findByPostID('post456');
```

### Complex Queries

```javascript
// Find waste posts with specific material
const plasticPosts = await WastePost.findByMaterialType('pet_bottles');

// Find forum posts by category
const tipsPosts = await ForumPost.findByCategory('Tips');

// Find applications by user
const userApplications = await Application.findByUserID('user123');
```

---

## 3. UPDATE (U) ✏️

### Static Update Method

```javascript
```

```javascript
const User = require('./models/User');

// Update user directly
const updatedUser = await User.update('user123', {
  firstName: 'Johnny',
  phone: '+9876543210',
  points: 150
});

console.log('Updated user:', updatedUser.firstName);
```

### Instance Update Method

javascript

```javascript
// Update through instance
const user = await User.findById('user123');
if (user) {
  await user.update({
    firstName: 'Johnny',
    status: 'Verified'
  });
}
```

### Specialized Update Methods

javascript

```javascript
// Add points to user
const user = await User.findById('user123');
await user.addPoints(50, 'Post_Creation');

// Update material price
const material = await Material.findByType('pet_bottles');
await material.updatePrice(25.50);

// Confirm a pickup
const pickup = await Pickup.findById('pickup456');
await pickup.confirm();

// Complete a pickup
await pickup.complete({
  itemName: 'PET Bottles',
  materialIDs: ['pet_bottles'],
  price: 75.50,
  kg: 3.0
}, 'proof-image-url');
```

---

## 4. DELETE (D) 🗑️

### Static Delete Method

javascript

```javascript
const User = require('./models/User');

// Delete user
const result = await User.delete('user123');
console.log(result.message); // "User deleted successfully"
```

### Instance Delete Method

javascript

```
// Delete through instance
const post = await Post.findById('post456');
if (post) {
  await post.delete();
  console.log('Post deleted');
}
```

### Soft Delete (Mark as inactive instead of actual deletion)

```javascript
// Instead of deleting, mark as inactive
const post = await Post.findById('post456');
await post.update({ status: 'Inactive' });

// Or for users
const user = await User.findById('user123');
await user.update({ status: 'Rejected' });
```

---

## Inheritance CRUD Operations

### How Inheritance Works in Our Setup

All post types (`WastePost`, `InitiativePost`, `ForumPost`) inherit from the base `Post` class and are stored in the same Firestore collection called `posts`.

#### The Magic: Single Collection, Multiple Types

```javascript
```

```
// All these are stored in the 'posts' collection:
{
  postID: "post1",
  postType: "Waste",      // Discriminator field
  title: "Plastic bottles",
  items: [...]            // Waste-specific field
}

{
  postID: "post2",
  postType: "Forum",      // Different type
  title: "Recycling tips",
  category: "Tips"        // Forum-specific field
}
```

---

## 1. CREATE with Inheritance ➕

### Create Specific Post Types

javascript

```javascript
const WastePost = require('./models/WastePost');
const InitiativePost = require('./models/InitiativePost');
const ForumPost = require('./models/ForumPost');

// Create waste post
const wastePost = await WastePost.create({
  userID: 'user123',
  title: 'Plastic bottles for pickup',
  description: 'I have 50 clean PET bottles',
  location: 'Manila, Philippines',
  items: [
    {
      itemName: 'PET Bottles',
      materialID: 'pet_bottles',
      sellingPrice: 15.50,
      kg: 2.5
    }
  ]
});

// Create initiative post
const initiativePost = await InitiativePost.create({
  userID: 'user456',
  title: 'Community Garden Project',
  description: 'Need materials for composting',
  materials: [
    {
      itemName: 'Cardboard boxes',
      materialID: 'boxes_cartons',
      kg: 10
    }
  ],
  projectDeadline: new Date('2024-12-31')
});

// Create forum post
const forumPost = await ForumPost.create({
  userID: 'user789',
  title: 'Best recycling tips?',
  description: 'What are your favorite recycling tips?',
  category: 'Tips'
});
```

**What Happens Behind the Scenes:**

```javascript
// All stored in same 'posts' collection like this:

// Firestore Document 1 (posts/wastepost123)
{
  postID: "wastepost123",
  postType: "Waste",        // Auto-set by WastePost
  userID: "user123",
  title: "Plastic bottles for pickup",
  description: "I have 50 clean PET bottles",
  location: "Manila, Philippines",
  status: "Active",
  items: [                  // Waste-specific field
    {
      itemName: "PET Bottles",
      materialID: "pet_bottles",
      sellingPrice: 15.50,
      kg: 2.5
    }
  ],
  createdAt: "2024-01-15T10:30:00Z"
}

// Firestore Document 2 (posts/forumpost456)
{
  postID: "forumpost456",
  postType: "Forum",        // Auto-set by ForumPost
  userID: "user789",
  title: "Best recycling tips?",
  description: "What are your favorite recycling tips?",
  status: "Active",
  category: "Tips",         // Forum-specific field
  createdAt: "2024-01-15T11:00:00Z"
}
```

---

## 2. READ with Inheritance 📖

### Read Specific Post Type

```javascript
// Find all waste posts
const wastePosts = await Post.findByType('Waste');
// Returns array of WastePost instances with .items field

// Find all forum posts
const forumPosts = await Post.findByType('Forum');
// Returns array of ForumPost instances with .category field

// Find specific waste post
const wastePost = await Post.findById('wastepost123');
// Returns WastePost instance (automatically detected from postType)
```

### Polymorphic Reading (Smart Type Detection)

```javascript
// The Post.findById method automatically returns the correct subclass:

const post = await Post.findById('somepost123');

if (post.postType === 'Waste') {
  console.log('Total value:', post.getTotalValue()); // WastePost method
  console.log('Materials:', post.getMaterialTypes()); // WastePost method
} else if (post.postType === 'Forum') {
  console.log('Category:', post.category); // ForumPost property
  const metrics = await post.getEngagementMetrics(); // ForumPost method
} else if (post.postType === 'Initiative') {
  console.log('Deadline:', post.projectDeadline); // InitiativePost property
  console.log('Is expired:', post.isExpired()); // InitiativePost method
}
```

### Inheritance-Specific Queries

```javascript
```

```javascript
// Waste post specific queries
const expensivePosts = await WastePost.findByPriceRange(20, 100);
const plasticPosts = await WastePost.findByMaterialType('pet_bottles');

// Forum post specific queries
const tipsPosts = await ForumPost.findByCategory('Tips');
const popularPosts = await ForumPost.getPopularPosts(10);

// Initiative post specific queries
const urgentProjects = await InitiativePost.findByDeadline(
  new Date(),
  new Date(Date.now() + 7 * 24 * 60 * 60 * 1000) // Next 7 days
);
```

---

## 3. UPDATE with Inheritance ✏️

### Update Base Post Fields

javascript

```javascript
// Update common fields (works for all post types)
const post = await Post.findById('post123');
await post.update({
  title: 'Updated title',
  description: 'New description',
  status: 'Inactive'
});
```

### Update Type-Specific Fields

javascript

```
// Update waste post items
const wastePost = await WastePost.findById('wastepost123');
await wastePost.addItem({
  itemName: 'Aluminum Cans',
  materialID: 'aluminum_cans',
  sellingPrice: 45.00,
  kg: 1.2
});

// Update forum post category
const forumPost = await ForumPost.findById('forumpost456');
await forumPost.updateCategory('Questions');

// Update initiative deadline
const initiativePost = await InitiativePost.findById('initiative789');
await initiativePost.updateDeadline(new Date('2024-06-30'));
```

### Smart Updates (Type-Aware)

```javascript
const post = await Post.findById('somepost123');

// The update works regardless of post type
await post.update({ status: 'Collected' });

// But type-specific methods only work on the right type
if (post instanceof WastePost) {
  const totalValue = post.getTotalValue(); // Only available on WastePost
} else if (post instanceof InitiativePost) {
  const isExpired = post.isExpired(); // Only available on InitiativePost
}
```

---

## 4. DELETE with Inheritance 🗑️

### Delete Any Post Type

```javascript
```

```javascript
// Delete works the same for all post types
const post = await Post.findById('post123');
await post.delete(); // Removes from 'posts' collection

// Or static method
await Post.delete('post123');
```

### Cascade Deletes (Handle Related Data)

javascript

```javascript
// When deleting a post, you might want to clean up related data
const post = await Post.findById('post123');

if (post) {
  // Delete related pickups
  const pickups = await Pickup.findByPostID(post.postID);
  await Promise.all(pickups.map(p => p.delete()));

  // Delete related comments
  const comments = await Comment.findByPostID(post.postID);
  await Promise.all(comments.map(c => c.delete()));

  // Delete related likes
  const likes = await Like.findByPostID(post.postID);
  await Promise.all(likes.map(l => l.delete()));

  // Finally delete the post
  await post.delete();
}
```

---

## Complete CRUD Examples 🔄

### Example 1: User Management

javascript

```javascript
const User = require('./models/User');

// CREATE
const user = await User.create({
  firstName: 'Alice',
  lastName: 'Johnson',
  email: 'alice@example.com',
  userType: 'Collector'
});

// READ
const foundUser = await User.findById(user.userID);
const collectors = await User.findByUserType('Collector');

// UPDATE
await foundUser.update({
  phone: '+1234567890',
  status: 'Verified'
});

// UPDATE with business logic
await foundUser.addPoints(25, 'Post_Creation');
await foundUser.addBadge('first_post_badge');

// DELETE
await foundUser.delete();
```

### Example 2: Post Management with Inheritance

javascript

```javascript
const WastePost = require('./models/WastePost');
const ForumPost = require('./models/ForumPost');

// CREATE different post types
const wastePost = await WastePost.create({
  userID: 'user123',
  title: 'Recyclable materials available',
  description: 'Various plastic items ready for collection',
  items: [
    { itemName: 'Water Bottles', materialID: 'pet_bottles', sellingPrice: 12, kg: 1.5 }
  ]
});

const forumPost = await ForumPost.create({
  userID: 'user456',
  title: 'Recycling tips for beginners',
  description: 'Share your best recycling advice',
  category: 'Tips'
});

// READ - Polymorphic reading
const anyPost = await Post.findById(wastePost.postID);
console.log('Post type:', anyPost.postType); // "Waste"

if (anyPost.postType === 'Waste') {
  console.log('Total value:', anyPost.getTotalValue()); // WastePost method
}

// READ - Type-specific queries
const allWastePosts = await Post.findByType('Waste');
const tipsPosts = await ForumPost.findByCategory('Tips');

// UPDATE - Common fields
await anyPost.update({
  title: 'Updated title',
  status: 'Inactive'
});

// UPDATE - Type-specific fields
if (anyPost.postType === 'Waste') {
  await anyPost.addItem({
    itemName: 'Plastic Bags',
    materialID: 'plastic_bags_sachets',
    sellingPrice: 8,
    kg: 0.5
  });
}

// DELETE
await anyPost.delete();
```

### Example 3: Complex Workflow

```
javascript
```

```
// Complete pickup workflow
const WastePost = require('./models/WastePost');
const Pickup = require('./models/Pickup');
const User = require('./models/User');

// 1. CREATE waste post
const wastePost = await WastePost.create({
  userID: 'giver123',
  title: 'Aluminum cans available',
  description: '20kg of clean aluminum cans',
  location: 'Quezon City',
  items: [
    { itemName: 'Aluminum Cans', materialID: 'aluminum_cans', sellingPrice: 45, kg: 20 }
  ]
});

// 2. CREATE pickup request
const pickup = await Pickup.create({
  postID: wastePost.postID,
  giverID: 'giver123',
  collectorID: 'collector456',
  pickupTime: new Date('2024-02-01T14:00:00Z'),
  pickupLocation: 'Quezon City'
});

// 3. READ and UPDATE pickup through workflow
const foundPickup = await Pickup.findById(pickup.pickupID);

// Confirm pickup
await foundPickup.confirm(); // Sets status to 'Confirmed', adds confirmedAt

// Complete pickup
await foundPickup.complete({
  itemName: 'Aluminum Cans',
  materialIDs: ['aluminum_cans'],
  price: 900, // 20kg * 45 per kg
  kg: 20
}, 'proof-of-pickup-url');

// 4. READ final state
const completedPickup = await Pickup.findById(pickup.pickupID);
console.log('Final status:', completedPickup.status); // "Completed"
```

```
// 5. UPDATE related post
await wastePost.update({ status: 'Collected' });
```

---

## Advanced CRUD Patterns 🎯

### Batch Operations

```javascript
// Create multiple related records
const createPickupWorkflow = async (postID, giverID, collectorID) => {
  // Create pickup
  const pickup = await Pickup.create({
    postID, giverID, collectorID,
    pickupTime: new Date(),
    pickupLocation: 'TBD'
  });

  // Create notification
  const notification = await Notification.create({
    userID: giverID,
    type: 'Pickup',
    title: 'New Pickup Request',
    message: 'Someone wants to collect your waste',
    referenceID: pickup.pickupID
  });

  return { pickup, notification };
};
```

### Transaction-like Operations

```javascript
```

```
// Update multiple related records together
const completePickupTransaction = async (pickupID, finalWasteData) => {
  try {
    const pickup = await Pickup.findById(pickupID);
    const post = await Post.findById(pickup.postID);

    // Update pickup
    await pickup.complete(finalWasteData);

    // Update post status
    await post.update({ status: 'Collected' });

    // Both giver and collector get points (handled in pickup.complete())

    return { success: true };
  } catch (error) {
    // If any step fails, you might want to rollback
    throw new Error(`Transaction failed: ${error.message}`);
  }
};
```

### Query Inheritance Hierarchy

```javascript
// Get all posts (any type)
const allPosts = await Post.findByUserID('user123');

// Filter by specific types
const userWastePosts = allPosts.filter(post => post.postType === 'Waste');
const userForumPosts = allPosts.filter(post => post.postType === 'Forum');

// Or use type-specific queries
const wastePostsOnly = await Post.findByType('Waste');
```

---

## Key Benefits of Our Approach ✨

### 1. Type Safety with Flexibility

```javascript
```

```
// You get the right class instance automatically
const post = await Post.findById(postID);
// post is automatically WastePost, ForumPost, or InitiativePost based on postType
```

### 2. Shared and Specific Methods

```javascript
// All post types can use base methods
await post.update({ status: 'Active' }); // Works for any post type

// But also have their own specific methods
if (post.postType === 'Waste') {
  const value = post.getTotalValue(); // Only WastePost has this
}
```

### 3. Single Source of Truth

```javascript
// All posts in one collection = easier queries
const recentPosts = await Post.findByUserID('user123'); // Gets ALL post types
const activePosts = await Post.findByStatus('Active');  // Gets ALL active posts
```

### 4. Clean Business Logic

```javascript
// Complex operations are simple method calls
await wastePost.markAsCollected('collector123');
await pickup.confirm();
await user.addPoints(50, 'Pickup_Completion');
```

This inheritance approach gives you the best of both worlds: **shared functionality** in the base `Post` class and **specialized features** in each subclass! 🎯