Agile Scrum Estimation Activity

Consider ecommerce application for agile scrum estimation activity.

1. Start with the Epics:

- User Registration
- User Login
- Product Search & Catalogue
- Cart

Important Scrum rule: You never estimate epics directly with Planning Poker.

You break epics → user stories → tasks, then estimate user stories.

2. Break Epics into User Stories + Tasks

Epic 1: User Registration

User Stories

US-1: As a user, I want to register using email & password so that I can create an account.

Tasks

- Design registration UI
- Validate input fields (email, password, confirm password)
- Backend API to create user
- Password hashing & storage
- Email uniqueness check
- Success & error handling
- Unit testing

Estimate (Planning Poker): 5 story points

US-2: As a user, I want to receive a confirmation after successful registration.

Tasks

- Trigger confirmation email
- Email template
- Integration with email service
- Test email flow

Estimate: 3 story points

Epic Total (Registration): 8 story points

Epic 2: User Login

User Stories

US-3: As a user, I want to log in using my credentials.

Tasks

- Login UI
- Input validation
- Backend authentication API
- Password verification
- Error handling
- Unit tests

Estimate: 5 story points

US-4: As a user, I want to stay logged in during my session.

Tasks

- Token/session handling
- Secure storage
- Session expiry logic
- Testing

Estimate: 3 story points

Epic Total (Login): 8 story points

Epic 3: Product Search & Catalogue

User Stories

US-5: As a user, I want to see a list of products.

Tasks

- Product listing UI
- Backend API for products
- Database schema
- Pagination
- Error handling

- Testing

Estimate: 8 story points

US-6: As a user, I want to search products by keyword.

Tasks

- Search UI
- Search API
- Database query optimization
- No-result handling
- Testing

Estimate: 5 story points

US-7: As a user, I want to view product details.

Tasks

- Product details UI
- API for product details
- Image handling
- Testing

Estimate: 5 story points

Epic Total (Product & Catalogue): 18 story points

Epic 4: Cart

User Stories

US-8: As a user, I want to add products to my cart.

Tasks

- Add-to-cart UI
- Cart backend logic
- Persist cart (session/db)
- Testing

Estimate: 5 story points

US-9: As a user, I want to view my cart.

Tasks

- Cart page UI
- Fetch cart API
- Price calculation
- Testing

Estimate: 5 story points

US-10: As a user, I want to update or remove items from cart.

Tasks

- Quantity update logic
- Remove item logic
- UI updates
- Testing

Estimate: 5 story points

Epic Total (Cart): 15 story points

3. How to Do Planning Poker (Practically)

Step-by-step in the session:

Read the user story

Team discusses:

1. Complexity
2. Unknowns
3. Integration effort

Each member chooses a card:

1, 2, 3, 5, 8, 13

- **Reveal simultaneously**
- **If big difference:**

- **Highest & lowest explain**

Re-vote

Final consensus = story point value

Rule of thumb

1–2 → very small

3–5 → medium

8 → complex

13 → too big → break story

4. Choosing MVP (Most Important Part)

MVP Goal:

User can browse products and add them to cart

MVP Must-Have Stories

Priority	User Story	Points
☑	View product list	8
☑	View product details	5
☑	Add to cart	5
☑	View cart	5
⚠️	Login	5
⚠️	Registration	5

👉 MVP Total ≈ 28–33 points

5. Sprint Planning Example

Assume:

Team velocity = 20 points / sprint

Sprint length = 2 weeks

Sprint 1 (MVP-focused)

Product list (8)

Product details (5)

Add to cart (5)

Total: 18 points

Sprint 2

View cart (5)

Login (5)

Registration (5)

Total: 15 points

After Sprint 2 → MVP release ready

6. Release Planning Summary

Sprint 1: Browsing capability

Sprint 2: Basic user + cart flow

Release 1 (MVP):

Users can browse products and add to cart (optionally logged in)

What is an API?

API (Application Programming Interface) is a way for two software systems to talk to each other.

Think of an API as a waiter in a restaurant:

- You (frontend / app) → place an order
- Waiter (API) → takes the request to the kitchen
- Kitchen (backend / database) → prepares the food
- Waiter (API) → brings the response back to you

You never go into the kitchen directly.

Why do we need APIs?

1. Separation of responsibilities

Frontend: UI (buttons, forms, pages)

Backend: Business logic + database

API: The bridge between them

This keeps systems clean, secure, and scalable.

2. Security

APIs control:

- What data can be accessed
- Who can access it
- What actions are allowed

Example:

Frontend cannot directly access database

API validates user, permissions, input

3. Reusability

One API can be used by:

Web app

Mobile app

Third-party integrations

Same backend, multiple clients.

4. Scalability

You can:

Change frontend without touching backend

Improve backend performance without breaking UI

APIs make systems flexible.

API in Your E-commerce Application (Very Practical)

Let’s map it to your epics.

1. **Registration API**

Frontend sends:

POST /api/register

```
{
  "email": "user@test.com",
```

"password": "123456"

}

Backend:

Validates input

Stores user

Returns success/failure

2. Login API

POST /api/login

{

"email": "user@test.com",

"password": "123456"

}

Returns:

{

"token": "abc123"

}

Token is used for secure access later.

3. Product Catalogue API

GET /api/products

Returns list of products.

Product Search API

GET /api/products?search=phone

Cart API

POST /api/cart/add

{

"productId": 101,

"quantity": 1

}