Crossbow Delivery Service

Link to the deployed application:

https://crossbowimg-616137423630.asia-south2.run.app

Just click on the link and get started! Too see detailed steps, please go to page 2

Setting up the application:

Step 1: Clone the Repository

Step 2: Build and Run with Docker Compose

docker-compose up --build

Step 3: Access the Application

Once all containers are running, open your web browser and go to: http://localhost:8080

Step 4: Stopping the Application

To stop the application, press `Ctrl+C` in the terminal where docker-compose is running, or open a new terminal and run: docker-compose down

Architecture Choices:

a) Framework:

- Django (5.2.1) as the main web framework
- Django REST Framework for API endpoints
- SQLite for development (with PostgreSQL configuration commented out for production)
- Redis as the message broker and result backend for Celery

b) Async Task/Queueing System:

- Celery for handling asynchronous webhook deliveries
- Redis as the message broker and result backend
- Exponential backoff retry strategy with 5 maximum retries
- Retry delays: 10s, 30s, 1m, 5m, 15m (progressive backoff)

c) Retry Strategy:

- Maximum of 5 retry attempts
- Progressive backoff with increasing delays
- Detailed logging of retry attempts and failures
- Automatic cleanup of old delivery logs after 72 hours

Database Schema and Indexing Strategies:

a) Models:

- Subscription Model:

  - Foreign key to User
  - Unique subscription_id
  - target_url for webhook delivery
  - Optional secret_key for signature verification
  - event_types as comma-separated list
  - Indexes on (user, subscription_id) and created_at

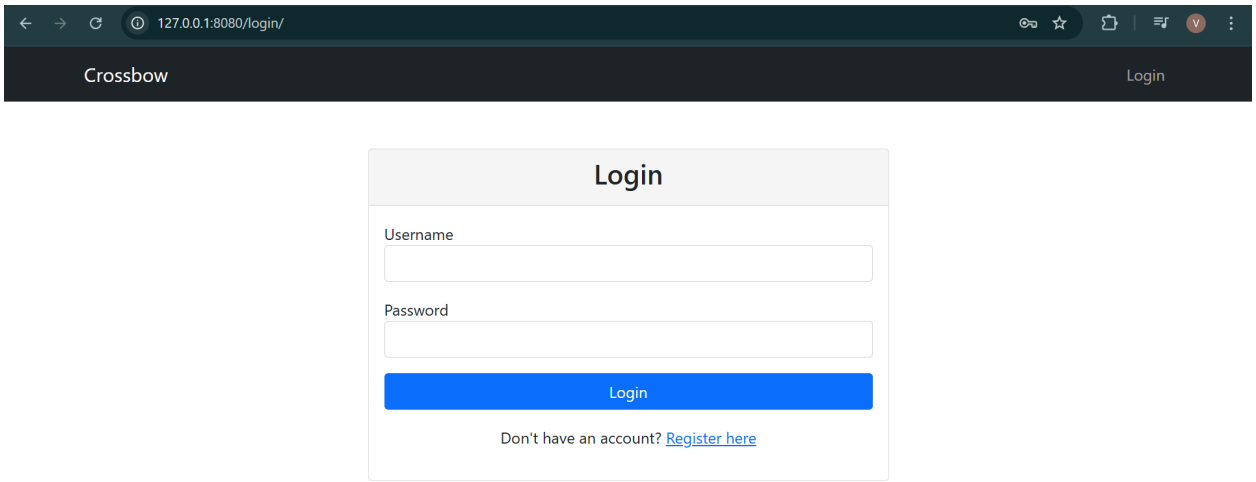
- DeliveryLog Model:
  - Foreign key to Subscription
  - attempt_number for tracking retries
  - status field for delivery state
  - http_status_code and error_details for debugging
  - event_type for tracking
  - Indexes on (subscription, timestamp), status, and event_type

b) Indexing Strategy:
- Composite indexes for common query patterns
- Timestamp-based indexing for log cleanup operations
- Status-based indexing for monitoring and analytics
- Event type indexing for filtering and reporting

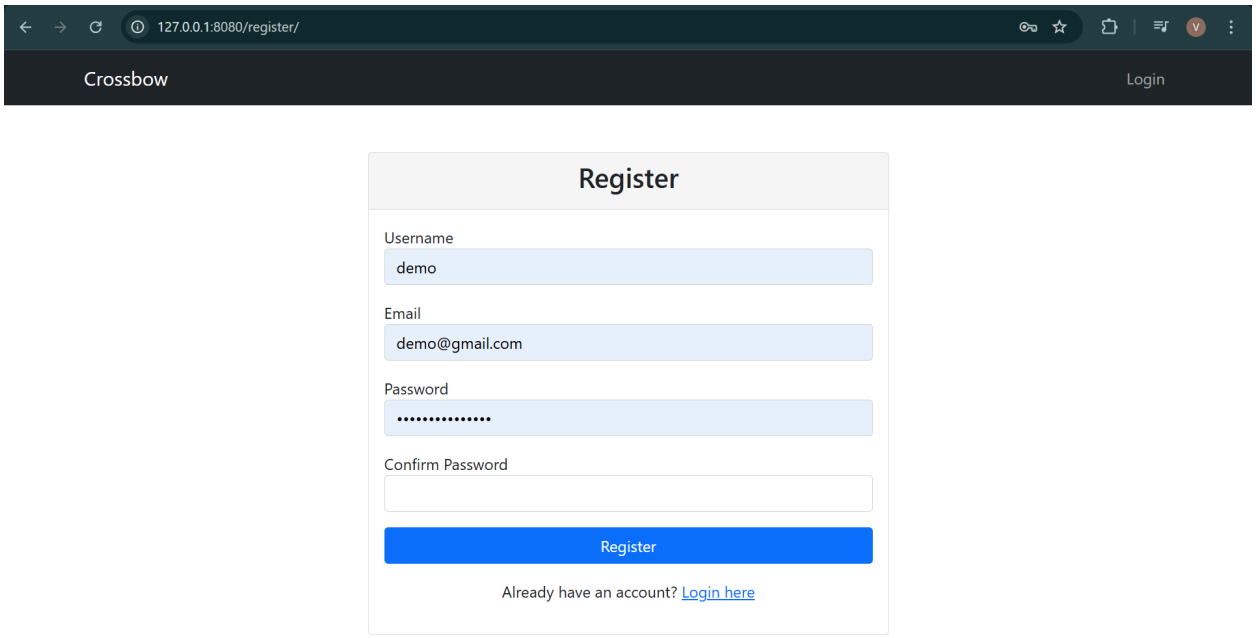
Using the application:

To see video explanation:
https://drive.google.com/drive/folders/19xUze09-YtprMSHmp77FV18-F736Bmio?usp=sharing

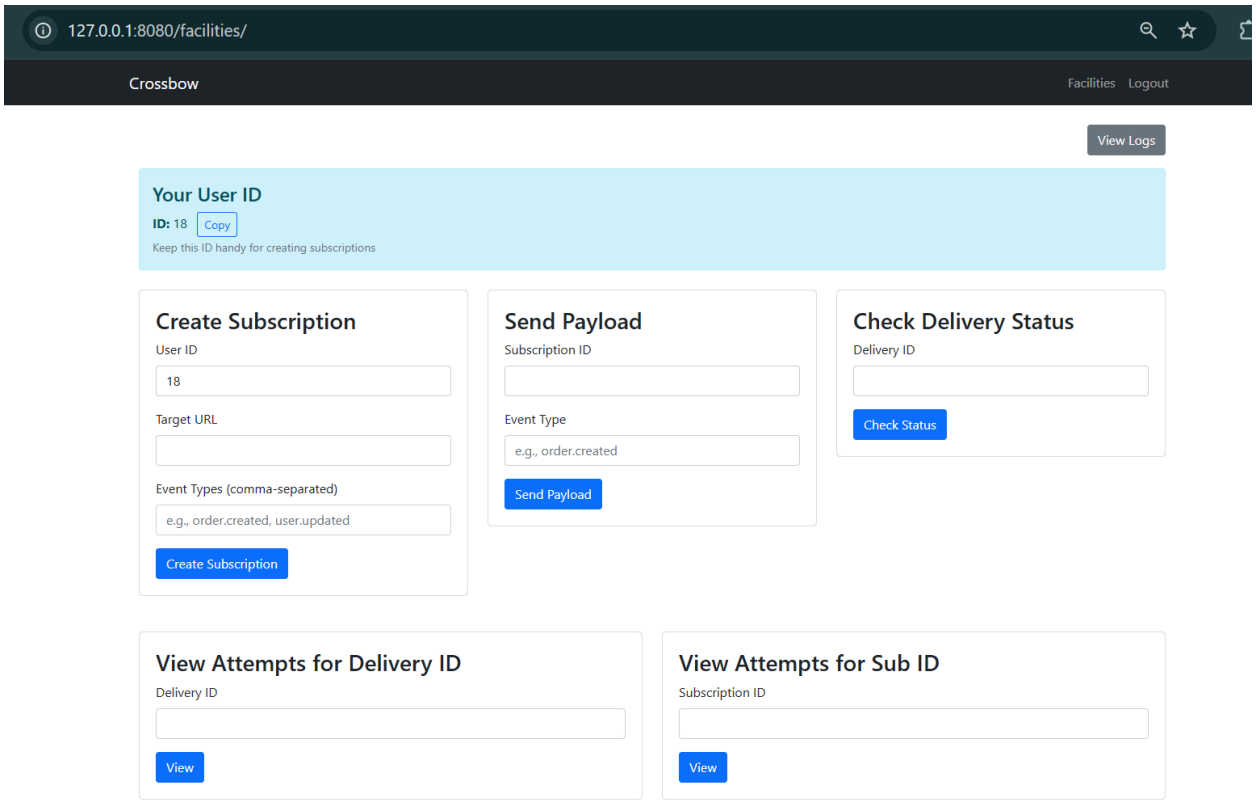
Step-1: Login and Registration:



If you do not have an account, go to the registration page and register your account

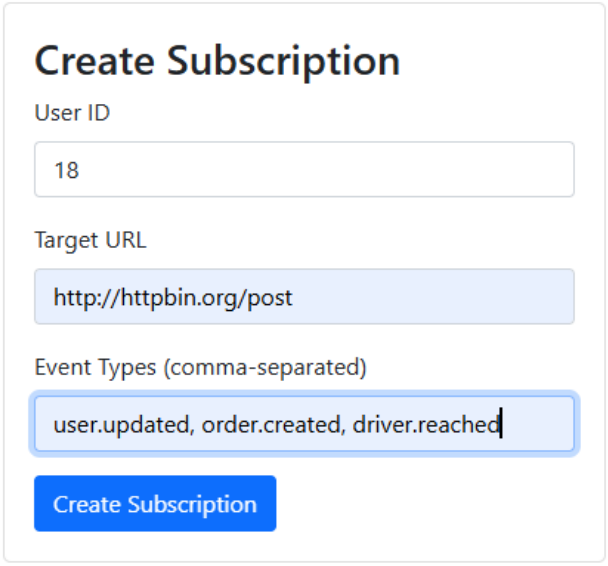


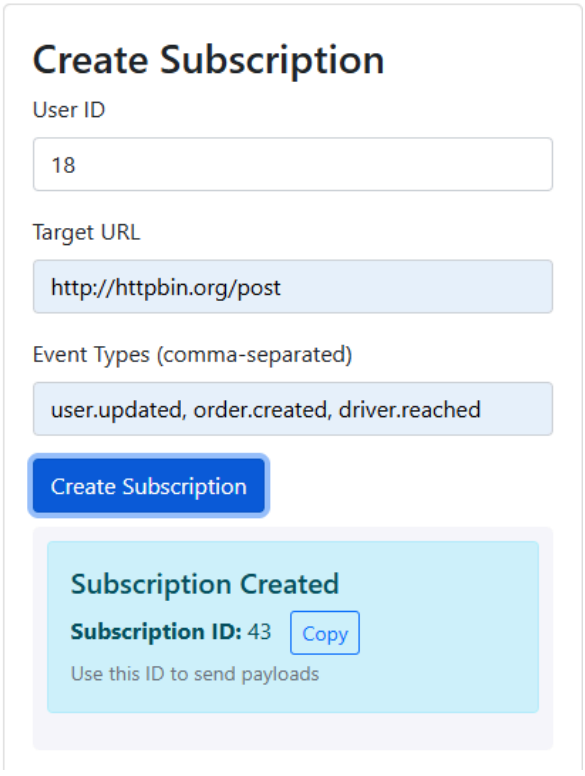
Once you register, you'll be automatically logged in



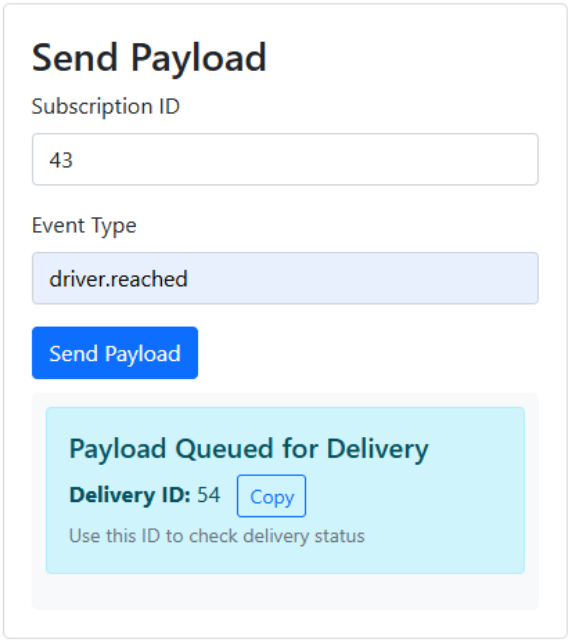
Step-2: Creating subscription
Copy your user id and create a subscription

You can write comma separated events for which you want to receive notifications

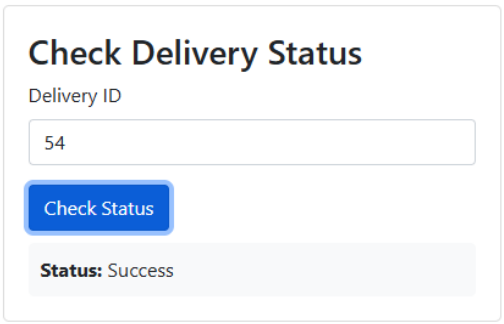




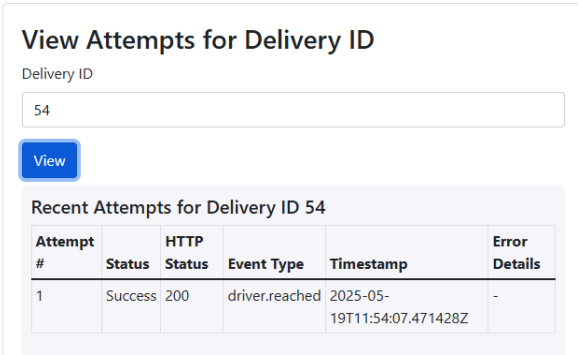
Step-2: Sending Payload:
Copy Subscription id and also a single event for which the notification is being sent.



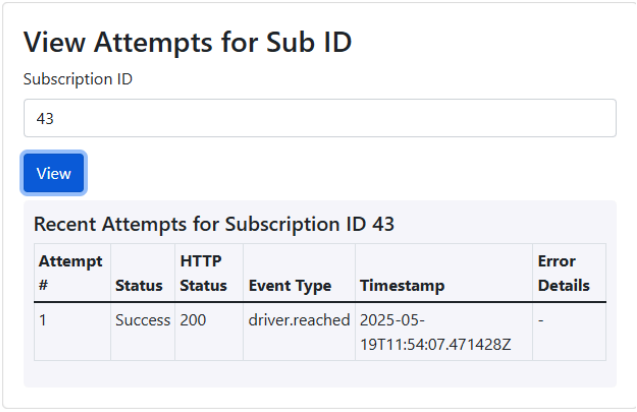
Step-3: Checking Delivery Status:
Copy Delivery id and enter



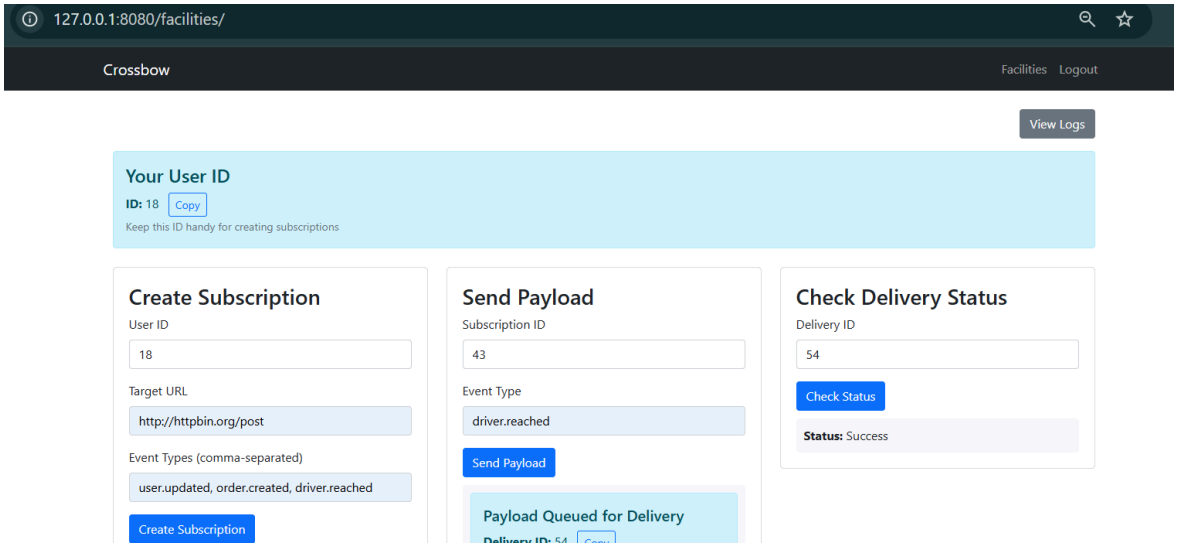
Step-4: View attempt for a specific Delivery id:
Copy Delivery id and enter

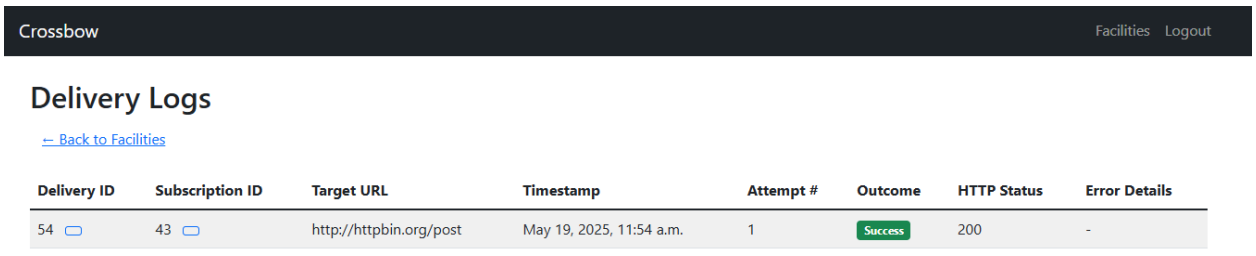


Step-5: View attempt for a specific Subscription id:
Copy Subscription id and enter

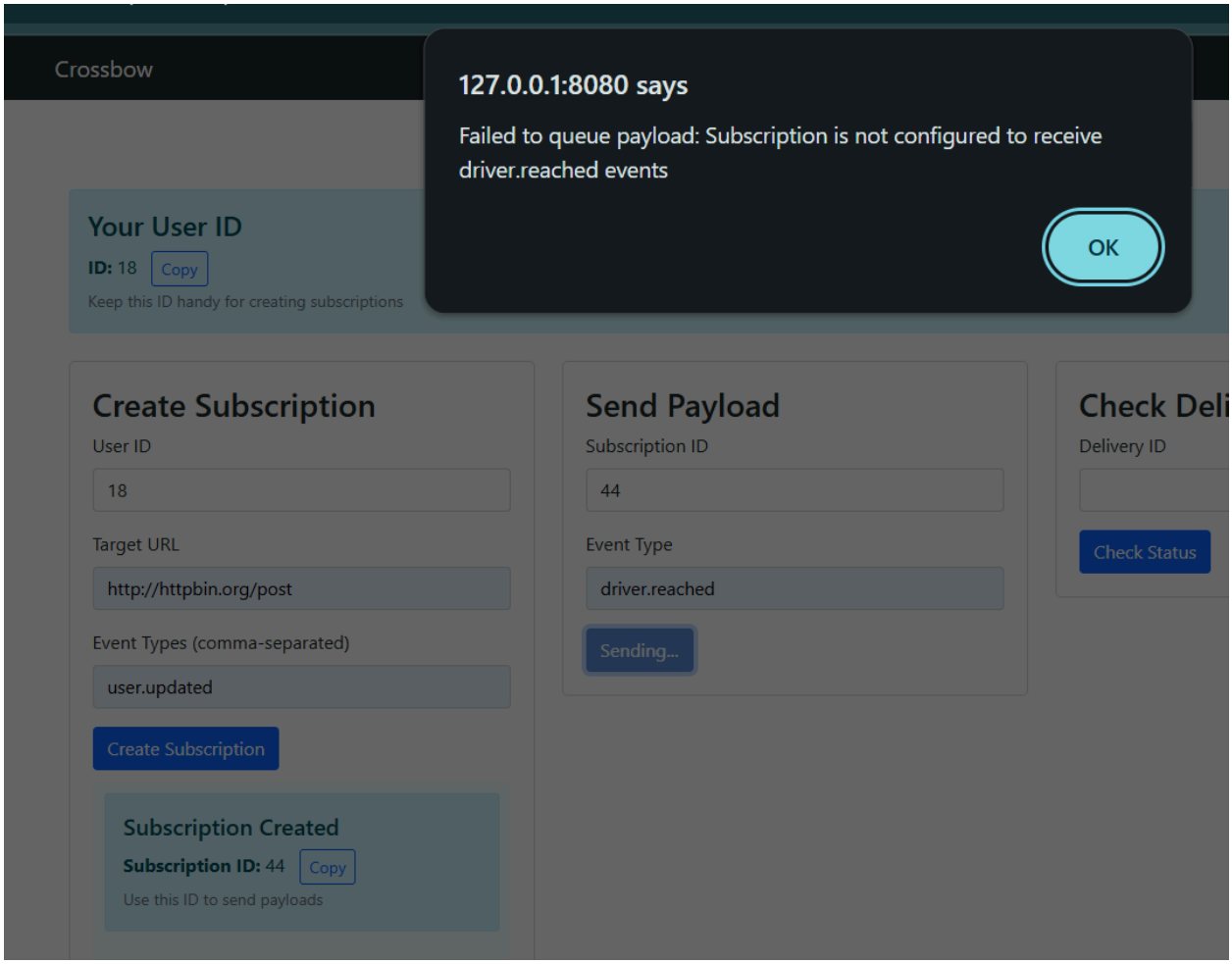


Step-6: View Logs:
Click on view logs button in top right corner





In case the event type for the payload is not supported by the subscription, user is shown an error message:



Monthly Cost Estimation for Cloud Run Free Tier:

Assumptions:
- 5000 webhooks/day = ~150,000 webhooks/month
- 1.2 delivery attempts per webhook = ~180,000 total attempts/month
- Average webhook size: 1KB
- Average processing time: 2 seconds per attempt

Cost Breakdown:
a) Cloud Run Free Tier:
- 2 million requests/month free
- 360,000 vCPU-seconds/month free
- 180,000 GiB-seconds/month free

Our usage:
- Requests: 180,000/month (well within free tier)
- vCPU-seconds: ~360,000/month (within free tier)
- Memory: ~180,000 GiB-seconds/month (within free tier)

Total Cost: $0 (within free tier limits)

Assumptions Made:

a) Infrastructure:
- Cloud Run is used as the deployment platform
- Redis is used for message queuing
- PostgreSQL would be used in production (currently SQLite in development)

b) Performance:
- Average webhook processing time of 2 seconds
- 1KB average payload size
- 72-hour retention period for delivery logs
- 5 maximum retry attempts with exponential backoff

c) Security:
- Secret key for webhook signature verification is optional
- CORS is currently set to allow all origins (development only)
- Session-based authentication is used

d) Scalability:
- The system can handle the specified load within Cloud Run free tier
- Redis can handle the message queue load
- Database indexes are optimized for the expected query patterns

The architecture is well-suited for a webhook delivery system with good reliability through retries, proper monitoring through logging, and efficient resource usage. The free tier of Cloud Run should be sufficient for the specified load, making it a cost-effective solution.