PROJECT DEVELOPMENT SUMMARY (Phase 2 → Phase 3.5)

Overview:

This document explains the overall backend development journey from Phase 2 to Phase 3.5. Each phase includes its purpose, main tasks, how it works in the project, steps to test, required commands/tools, and flow explanation. This structure ensures you clearly understand what each part is contributing toward the final goal of WhatsApp-based customer call automation.

■■

PHASE 2 – BACKEND SKELETON SETUP

■■

✓ Objective:

Establish a clean backend foundation using FastAPI, SQLAlchemy, PostgreSQL, Alembic, JWT Authentication, and modular structure.

✓ What was done:

• FastAPI project structure created.

• Database connection using SQLAlchemy configured.

• Dependencies setup (session handling, authentication).

• Basic API routes & JWT authentication added.

• Alembic installed & database version control enabled.

✓ Important files:

• app/main.py – Application entry point.

• app/core/database.py – DB connection.

• app/core/deps.py – Dependency injection.

• app/api/v1/endpoints/auth.py – Login authentication.

• alembic/env.py – Migration configuration.

✓ Commands used:

1. alembic init alembic
2. alembic revision --autogenerate -m "initial"
3. alembic upgrade head
4. pip install fastapi uvicorn sqlalchemy psycopg2-binary alembic passlib python-jose

✓ How to test:

1. Run API → uvicorn app.main:app --reload
2. Open Swagger → http://127.0.0.1:8000/docs
3. Test login via → POST /api/v1/auth/login

Use example credentials (created earlier)

■■

PHASE 3.1 – WHATSAPP CLIENT DEVELOPMENT

■■

✓ Objective:

Build a client to send WhatsApp text messages using Meta WhatsApp Cloud API.

✓ What was done:

• WhatsApp client class created.

• send_text_message() method developed.

• Environment variables added:

WHATSAPP_ACCESS_TOKEN

WHATSAPP_PHONE_NUMBER_ID

WHATSAPP_FROM_NUMBER

✓ Important files:

• app/services/whatsapp_client.py

• app/api/v1/endpoints/whatsapp_test.py

✓ Testing:

POST /api/v1/whatsapp-test/test

Body:

```
{
"to": "whatsapp:+91xxxxxx",
"body": "Test message"
}
```

Observe terminal logs for confirmation.

■■

PHASE 3.2 – CELERY INTEGRATION

■■

✓ Objective:

Move WhatsApp sending to asynchronous background tasks.

✓ What was done:

• Celery + Redis added.

• Task queue created.

• Celery worker configured.

✓ Commands:

```
redis-server
celery -A app.tasks.celery_app worker --loglevel=info
```

✓ Important files:

• app/tasks/celery_app.py

• app/tasks/whatsapp_tasks.py

✓ Test:

Send WhatsApp test and observe Celery logs:

```
Task whatsapp.send_text[...] received
```

■■■

PHASE 3.3 – DATABASE MODELS

■■■

✓ Objective:

Store WhatsApp automation settings per tenant.

✓ What was done:

• Created TenantSettings model.

• Fields: tenant_id, enabled, min_call_duration_seconds.

• Created migration.

✓ Commands:

```
alembic revision --autogenerate -m "create tenant settings"
alembic upgrade head
```

✓ Test:

Insert values into tenant_settings table in DB.

■■■

PHASE 3.4 – AUTOMATION SERVICE & TASK QUEUING

■■■

✓ Objective:

Trigger a Celery WhatsApp task programmatically upon valid events.

✓ What was done:

• handle_call_automation() function written.

• It calls send_whatsapp_text_task.delay(...)

• Improved logging.

✓ Important files:

• app/services/automation_service.py

✓ Test (manual call):

```
from_number = "+91xxxxxxxx"
to_number = "+91xxxxxxxx"
call_id = 1
tenant_id = 1
```

Call handle_call_automation() from API or Python shell.

■■

PHASE 3.5 – CALL WEBHOOK HANDLING + AUTOMATION TRIGGER

■■

✓ Objective:

Receive inbound call events → Save call details → Check conditions → Trigger WhatsApp message using Celery.

✓ Steps followed:

1. Endpoint added → POST /api/v1/webhooks/calls/{tenant_slug}
2. Request validated using WebhookCallRequest schema.
3. WebhookCall record stored in DB.
4. Conditions evaluated:

• settings.enabled == True

• call_duration_seconds ≥ min_call_duration_seconds

• status in [completed, answered, done]

5. If passed → WhatsApp task queued.

✓ Example JSON:

```
{
"direction": "inbound",
"from_number": "+911234567890",
"to_number": "+919999999999",
"customer_number": "+911234567890",
"status": "completed",
"provider": "sandbox",
"provider_call_id": "CALL123",
"duration_seconds": 45,
"started_at": "2025-11-26T09:00:00Z",
"ended_at": "2025-11-26T09:00:45Z",
"raw_payload": { "note": "test call for automation" }
}
```

✓ Successful test indicators:

• Response shows "automation_triggered": true

• Celery logs:

```
Task whatsapp.send_text[...] received
■ Celery Task Triggered → Sending WhatsApp
```

■■

SYSTEM FLOW OVERVIEW (Phase 3.5 FINAL)

■■

Incoming Webhook →

Validate Tenant →

Save Call Record →

Check Automation Rules →

Queue Celery Task →

Send WhatsApp Message

■■■

NEXT STEPS

■■■

→ Begin Phase 3.6: Add retry mechanism, logging enhancement, and real-time progress tracking.