# Backend API Runbooks Pack — TL;DR Quick Start

| Step | Action / Notes |
|---|---|
| Install Cursor | Download from cursor.com for macOS/Windows/Linux |
| Open Repo | Open your repository folder in the Cursor IDE |
| Optional CLI | Set up the optional Cursor CLI (`cursor-agent`) for command-line workflows |
| Generate Endpoint | Enter a Cursor prompt in IDE or CLI (example: `Generate a Python FastAPI POST /auth/login endpoint with JWT & logging`) |
| Test | Run generated code with your test runner (e.g., `pytest tests/`) |
| Apply Templates / Runbooks | Store your prompt in templates or runbooks, then execute via Cursor |

> **Note:** `cursor generate` / `cursor run` are shorthand for prompts executed inside the Cursor IDE or Agent. Not pip/npm commands.

---

## What is Cursor & Why It Matters for Backend Engineers

Cursor is a code-first AI tool that streamlines backend API development. It reduces repetitive coding, automates boilerplate, and enforces consistent project structure. Engineers focus on logic, making development faster and less error-prone.

### Cursor's Role in Backend Development

- Automates repetitive tasks: endpoints, logging, database integration
- Generates structured, production-ready scripts from prompts
- Saves hours writing boilerplate, validation, and tests

**Example Prompt:**

| Prompt |
|---|
| Generate a Python REST API endpoint for user login with JWT and error logging |

**Example Generated Code:**

```
@router.post("/auth/login")
async def login(payload: LoginRequest):
    user = await UserService.authenticate(payload.email, payload.password)
```

```
    if not user:
        raise HTTPException(status_code=401, detail="Invalid credentials")
    token = jwt.create_access_token({"sub": str(user.id)})
    logger.info(f"Login attempt by {payload.email}")
    return {"access_token": token, "token_type": "bearer"}
```

---

## Typical Backend API Challenges

| Challenge | How Cursor Helps |
|---|---|
| Slow development | Generate endpoints in minutes |
| Debugging issues | Built-in validation & logging |
| Inconsistent endpoints | Standardized templates |
| Limited automation | Generate tests, logs, and docs |

---

## Benefits of a Code-First Approach

- **Speed:** Endpoints ready in minutes
- **Consistency:** Standardized naming, logging, DB integrations
- **Error Reduction:** Auto-validation, structured prompts, tests
- **Scalability:** Multi-endpoint projects without extra boilerplate

---

## 1. Set Up Cursor for Backend API Project

### Install Cursor

| Step | Notes |
|---|---|
| IDE | Download from cursor.com |
| CLI (Optional) | Install `cursor-agent` for terminal prompts |
| Open Project | Open repo folder in Cursor IDE |

### Link Repository

| Step | Notes |
|---|---|
| Connect | Link GitHub/GitLab account in Cursor IDE |
| Authenticate | Select repository and grant write permissions |

| Step | Notes |
|---|---|
| Tip | Use a dedicated branch for generated scripts |

### Add Language & Framework

| Stack | Notes |
|---|---|
| Python | FastAPI, Flask, Django |
| Node.js | Express.js, NestJS |
| Database | PostgreSQL, MySQL, MongoDB |
| Steps | Configure language → Add framework → Connect DB |

### Verify Setup with Sample Project

| Command / Prompt | Notes |
|---|---|
| `mkdir cursor-test && cd cursor-test` | Create test folder |
| `Generate FastAPI endpoint /status returning app version and uptime` | Cursor generates endpoint |
| `uvicorn app:app --reload` | Run server locally |

**Performance Improvement Example:**

| Task | Manual Time | With Cursor |
|---|---|---|
| Initialize project | 35–45 min | 8–10 min |
| First working endpoint | 60–90 min | 12–15 min |
| Basic logging setup | 20–30 min | 4–5 min |

---

## 2. Create Your First Cursor Backend API

### Define API Endpoint

| Route | Method | Inputs | Outputs |
|---|---|---|---|
| /auth/login | POST | email, password | JWT token, user ID, status |

### Cursor Prompt Example

| Prompt |
| --- |
| Generate a Python FastAPI POST endpoint /auth/login that validates user credentials, returns a JWT token, logs attempts, and handles invalid logins gracefully. |

### Review Generated Code

- Validate inputs
- Check logging
- Verify JWT generation
- Ensure error handling

**Mini-Checklist:**

-

### Test Endpoint Locally

| Command / Prompt | Notes |
| --- | --- |
| `uvicorn app:app --reload` | Start FastAPI server |
| `curl -X POST http://localhost:8000/auth/login -H "Content-Type: application/json" -d '{"email":"test@example.com","password":"pass123"}'` | Test login |

## 3. Advanced Backend API Workflows

### Generate Multiple Endpoints

| Prompt |
| --- |
| Generate Python FastAPI CRUD endpoints for /users with routes for create, read, update, and delete. Include input validation, logging, and error handling. |

### Database Integration Example

**SQLAlchemy Model:**

```
class Task(Base):
    __tablename__ = "tasks"
```

```
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    completed = Column(Boolean, default=False)
```

**CRUD Endpoint:**

```
@router.post("/tasks")
def create_task(title: str, db: Session = Depends(get_db)):
    task = Task(title=title)
    db.add(task)
    db.commit()
    db.refresh(task)
    return task
```

**Alembic Migration:**

```
alembic init alembic
alembic revision --autogenerate -m "create tasks table"
alembic upgrade head
```

### Schema Governance

```
from pydantic import BaseModel, EmailStr, constr

class UserCreate(BaseModel):
    email: EmailStr
    password: constr(min_length=8)

    class Config:
        extra = "forbid"
```

### Automated Unit Tests

| Prompt |
| --- |
| Generate Python pytest scripts for /auth/login endpoint. Include tests for valid login, invalid login, and missing fields. |

**Example pytest snippet:**

```
def test_login_success(client, test_user):
    resp = client.post("/auth/login", json={"email": test_user.email,
"password": "correct"})
```

```
    assert resp.status_code == 200
    assert "access_token" in resp.json()
```

### Logging Setup

| Prompt |
| --- |
| Generate Python FastAPI logging setup for /users endpoints. Include info, warning, and error logs for all CRUD operations. |

---

## 4. Use Templates & Runbooks

### Runbook Example: Add a New CRUD Module

| Cursor Prompt |
| --- |
| Runbook: Add JWT authentication to this backend. |

**Steps:**

1. Create /auth/login & /auth/verify routes
2. Add Pydantic models: LoginRequest, LoginResponse
3. Generate JWT utility
4. Add tests

### Templates Example

| Cursor Prompt |
| --- |
| Save this as template: "Generate CRUD endpoints for any resource with models, routing, logging, and tests." |

| Usage |
| --- |
| `cursor generate "Use CRUD template for resource: orders"` |

---

## Recommended Project Structure

```
backend/
  api/
    routers/
    controllers/
    schemas/
    services/
```

```
core/
  config/
  security/
  logging/
tests/
  integration/
  unit/
runbooks/
infra/
  docker/
  migrations/
```

---

## Common Pitfalls & Best Practices

1. **Keep Prompts Specific** – include framework, file paths, models, and behavior.
2. **Test Code Before Production** – run pytest locally.
3. **Version Control Generated Scripts** – diff before overwriting.
4. **Maintain Consistent Naming** – routes, models, and services predictable.
5. **Document API Endpoints** – auto-generate Markdown API reference using Cursor.