Hospital Management System - Troubleshooting Guide

This guide helps you diagnose and resolve common issues you might encounter while developing, testing, or deploying the Hospital Management System.

Table of Contents

---

1. Installation & Setup Issues

Problem: “Module not found” or “No module named ‘flask’“

Symptoms:

```
ModuleNotFoundError: No module named 'flask'
ImportError: No module named 'flask_sqlalchemy'
```

Causes:
- Virtual environment not activated
- Dependencies not installed
- Wrong Python environment

Solutions:

1. **Activate virtual environment:**
   ```bash
   # Windows
   venv\Scripts\activate
   ```

```
# macOS/Linux
source venv/bin/activate
```

1. **Install/Reinstall requirements:**
   ```bash
   pip install -r requirements.txt
   ```
2. **Verify installation:**
   ```bash
   pip list | grep Flask
   ```
3. **Upgrade pip if installation fails:**
   ```bash
   pip install --upgrade pip
   pip install -r requirements.txt
   ```
4. **Check Python path:**
   ```bash
   which python  # macOS/Linux
   where python  # Windows
   ```

Problem: “pip: command not found”

Symptoms:

```
bash: pip: command not found
```

Solutions:

1. **Use pip3 instead:**
   ```bash
   pip3 install -r requirements.txt
   ```
2. **Install pip:**
   ```bash
   # macOS/Linux
   python -m ensurepip –upgrade
   ```

```
# Windows
python -m ensurepip –upgrade
```

1. **Use python -m pip:**
   ```bash
   python -m pip install -r requirements.txt
   ```

Problem: “Permission denied” during installation

Symptoms:

```
PermissionError: [Errno 13] Permission denied
```

Solutions:

1. **Create and use virtual environment (Recommended):**
   ```bash
   python -m venv venv
   source venv/bin/activate  # or venv\Scripts\activate on Windows
   pip install -r requirements.txt
   ```

2. **Use –user flag (Not recommended for this project):**
   ```bash
   pip install --user -r requirements.txt
   ```

Problem: “Python version mismatch”

Symptoms:

```
Python 2.7 detected. This application requires Python 3.8+
SyntaxError: invalid syntax (f-strings not supported)
```

Solutions:

1. **Check Python version:**
   ```bash
   python --version
   python3 --version
   ```

2. **Use Python 3 explicitly:**
   ```bash
   ```

```
   python3 -m venv venv
   python3 app.py
```

3. **Install Python 3.8+:**
   - Download from python.org (https://www.python.org/downloads/)
   - Make sure to check “Add Python to PATH” during installation

---

2. Database Issues

Problem: “Database is locked”

Symptoms:

```
sqlite3.OperationalError: database is locked
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) database is
locked
```

Causes:
- Multiple processes accessing the database
- Previous Flask instance still running
- DB Browser for SQLite open
- File system issue

Solutions:

1. **Stop all Flask processes:**
   ```bash
    #  Find Flask processes
   ps aux | grep python  #  macOS/Linux
   tasklist | findstr python  #  Windows

 #  Kill the process (replace PID with actual process ID)
kill -9 PID  #  macOS/Linux
taskkill /PID PID /F  #  Windows
```

1. **Close DB Browser or any SQLite tool:**
   - Close any application that might have `hospital.db` open

2. **Delete lock file (if exists):**
   ```bash
   rm hospital.db-journal  # macOS/Linux
   del hospital.db-journal  # Windows
```

3. **Restart the application:**
   ```bash
   python app.py
   ```

4. **Last resort - Reset database:**
   ```bash
   # Backup first!
   cp hospital.db hospital_backup.db  # macOS/Linux
   copy hospital.db hospital_backup.db  # Windows

   # Delete and recreate
   rm hospital.db
   python app.py
   ```

Problem: "No such table" error

Symptoms:

```
sqlite3.OperationalError: no such table: user
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) no such
table: appointment
```

Causes:
- Database not initialized
- Database file deleted or corrupted
- Tables not created

Solutions:

1. **Check if database file exists:**
   ```bash
   ls -lh hospital.db  # macOS/Linux
   dir hospital.db     # Windows
   ```

2. **Delete and recreate database:**
   ```bash
   rm hospital.db  # macOS/Linux
   del hospital.db  # Windows
   ```

python app.py # This will recreate the database
```

1. **Force database initialization:**
   - Open `app.py` and ensure database initialization code runs:
   ```
   python
   with app.app_context():
       db.create_all()
   ```

2. **Check models are imported:**
   - Ensure all models are imported in `app.py` before `db.create_all()`

---

### Problem: "Database schema mismatch"

**Symptoms:**

```
sqlalchemy.exc.OperationalError: no such column: user.is_active
```

Causes:
- Database schema changed after creation
- Old database with outdated schema

Solutions:

1. **Reset database:**
   ```
   bash
   rm hospital.db
   python app.py
   ```

2. **Use migrations (Advanced):**
   ```
   bash
   pip install Flask-Migrate
   # Add migration code to app.py
   flask db init
   flask db migrate
   flask db upgrade
   ```

Problem: "Admin user not created"

Symptoms:
- Cannot login as admin
- Admin credentials don't work
   ```

Solutions:

1. **Check if admin exists in database:**
   ```
   python
   # Run in Python shell
   from app import app, db, User
   with app.app_context():
       admin = User.query.filter_by(username='admin').first()
       print(admin)
   ```

2. **Manually create admin:**
   ```python
   from app import app, db, User
   from werkzeug.security import generate_password_hash

with app.app_context():
admin = User(
username='admin',
email='admin@hospital.com',
password=generate_password_hash('admin 1 2 3'),
role='admin'
)
db.session.add(admin)
db.session.commit()
print("Admin created successfully!")
```

1. **Reset database:**
   ```
   bash
   rm hospital.db
   python app.py
   ```

---

3. Application Runtime Errors

Problem: "Address already in use" / "Port 5000 already in use"

Symptoms:

```
OSError: [Errno 48] Address already in use
OSError: [Errno 98] Address already in use
```

Solutions:

1. **Find and kill process using port 5000:**

macOS/Linux:
```bash
# Find process
lsof -i :5000

# Kill process
kill -9
```

Windows (Command Prompt as Administrator):
```bash
# Find process
netstat -ano | findstr :5000

# Kill process (replace PID with actual process ID)
taskkill /PID /F
```

1. **Use a different port:**

Edit `app.py`:
`python`
`if __name__ == '__main__':`
`app.run(debug=True, port=5001) # Change to 5001 or any available port`

1. **Restart your computer:**
   - As a last resort, restart to clear all processes

Problem: “Template not found”

Symptoms:

```
jinja2.exceptions.TemplateNotFound: admin/dashboard.html
```

Causes:
- Template file missing
- Wrong template path
- Incorrect folder structure

Solutions:

1. **Check template exists:**
   ```
   bash
     ls templates/admin/dashboard.html  # macOS/Linux
     dir templates\admin\dashboard.html  # Windows
   ```

2. **Verify folder structure:**
   ```
   templates/
     ├── admin/
     │   ├── dashboard.html
     │   └── ...
     ├── doctor/
     │   └── ...
     └── patient/
         └── ...
   ```

3. **Check route rendering code:**
   ```
   python
     return render_template('admin/dashboard.html')  # Correct path
   ```

4. **Case sensitivity matters on Linux/macOS:**
   - `Dashboard.html` ≠ `dashboard.html`

Problem: “Static files not loading (CSS/JS)”

Symptoms:
- No styling on pages
- Browser console shows `404` errors for CSS/JS files

Solutions:

1. **Check static folder structure:**
   ```
   static/
     ├── css/
     │   └── style.css
     ├── js/
     │   └── script.js
     └── images/
         └── ...
   ```

2. **Use correct Flask URL generation:**
   ```html
   ```

```

1. **Clear browser cache:**
   - Chrome: Ctrl+Shift+Delete (Cmd+Shift+Delete on Mac)
   - Or use Incognito/Private mode

2. **Check file permissions:**
   ```bash
   chmod -R 755 static/  # macOS/Linux
```

Problem: "Internal Server Error (500)"

Symptoms:
- Page shows "Internal Server Error"
- Application crashes with stack trace

Solutions:

1. **Enable debug mode:**
   ```python
   app.run(debug=True)
```

2. **Check terminal logs:**
   - Look at the Flask terminal for detailed error messages

3. **Common causes:**
   - Database query errors
   - Missing variables in template
   - Incorrect function arguments
   - Division by zero or NoneType errors

4. **Add try-except blocks:**
   ```python
   try:
       # Your code
   except Exception as e:
       print(f"Error: {e}")
       return "An error occurred", 500
```

5. **Check browser console:**
   - Press F12 to open Developer Tools
   - Look for JavaScript errors
```

4. Login & Authentication Issues

Problem: “Login not working” / “Invalid credentials” every time

Symptoms:
- Correct username/password shows “Invalid credentials”
- Cannot login to any account

Solutions:

1. **Verify user exists in database:**
   ```
   python
   from app import app, db, User
   with app.app_context():
       users = User.query.all()
       for user in users:
           print(f"{user.username} - {user.role}")
   ```

2. **Check password hashing:**
   ```python
   from werkzeug.security import check_password_hash
   from app import app, User

with app.app_context():
user = User.query.filter_by(username=’admin’).first()
print(check_password_hash(user.password, ‘admin 1 2 3 ’)) # Should print True
```

1. **Reset admin password:**
   ```python
   from app import app, db, User
   from werkzeug.security import generate_password_hash

with app.app_context():
admin = User.query.filter_by(username=’admin’).first()
admin.password = generate_password_hash(‘admin 1 2 3 ’)
db.session.commit()
print(“Password reset successfully!”)
```

1. **Check login route logic:**
   - Ensure `check_password_hash()` is used correctly
   - Verify form field names match POST data

Problem: “Session not persisting” / “Logged out immediately”

Symptoms:
- Login successful but redirects to login again
- Session doesn’t persist across pages

Solutions:

1. **Check SECRET_KEY is set:**
   ```
   python
   # In config.py or app.py
   app.config['SECRET_KEY'] = 'your-secret-key-here'
   ```

2. **Use Flask-Login properly:**
   ```python
   from flask_login import login_user, current_user, logout_user

@app.route(‘/login’, methods=[‘POST’])
def login():
 # … authentication logic …
login_user(user) # Don’t forget this!
return redirect(url_for(‘dashboard’))
```

1. **Check cookies are enabled in browser:**
   - Enable cookies in browser settings

2. **Use @login_required decorator:**
   ```python
   from flask_login import login_required

@app.route(‘/dashboard’)
@login_required
def dashboard():
return render_template(‘dashboard.html’)
```

Problem: “Accessing wrong portal after login”

Symptoms:
- Admin can access patient portal
- Patient can access admin portal

Solutions:

1. **Add role-based access control:**
   ```python
   from functools import wraps
   from flask_login import current_user
   from flask import abort

def admin_required(f):
@wraps(f)
def decorated_function(*args, kwargs):*
*if not current_user.is_authenticated or current_user.role != 'admin':*
*abort(403) # Forbidden*
*return f(*args, **kwargs)
return decorated_function

@app.route('/admin/dashboard')
@admin_required
def admin_dashboard():
return render_template('admin/dashboard.html')
```

1. **Redirect based on role after login:**
   ```python
   if user.role == 'admin':
       return redirect(url_for('admin_dashboard'))
   elif user.role == 'doctor':
       return redirect(url_for('doctor_dashboard'))
   elif user.role == 'patient':
       return redirect(url_for('patient_dashboard'))
   ```

---

5. Email Configuration Issues

Problem: "Email not sending"

Symptoms:
- No emails received after booking appointment
- No error messages but emails don't arrive

Solutions:

1. **Check .env file exists and is correct:**
   ```bash
   ```

```
cat .env  # macOS/Linux
type .env  # Windows
```

2. **Verify SMTP settings:**

```
env
MAIL_SERVER=smtp.gmail.com
MAIL_PORT=587
MAIL_USE_TLS=True
MAIL_USERNAME=your-email@gmail.com
MAIL_PASSWORD=your-app-password  # NOT your Gmail password!
```

3. **Test SMTP connection manually:**

```python
import smtplib

try:
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login('your-email@gmail.com', 'your-app-password')
print("✅ SMTP connection successful!")
server.quit()
except Exception as e:
print(f"❌ SMTP connection failed: {e}")
```

1. **Check Gmail App Password:**
   - Go to Google App Passwords (https://myaccount.google.com/apppasswords)
   - Generate new 16-character app password
   - Update `.env` file (remove spaces from app password)

2. **Enable 2-Step Verification:**
   - App Passwords require 2-Step Verification to be enabled

3. **Check spam/junk folder:**
   - Emails might be going to spam

Problem: "SMTPAuthenticationError"

Symptoms:

```
smtplib.SMTPAuthenticationError: (535, b'5.7.8 Username and Password not accepted')
```

Solutions:

1. **Use App Password, not regular Gmail password:**
   - Generate App Password at Google App Passwords (https://myaccount.google.com/apppasswords)
2. **Enable 2-Step Verification:**
   - Required for App Passwords
3. **Check credentials are correct:**
   - No typos in email or app password
   - Remove spaces from app password in `.env`
4. **For non-Gmail providers:**
   - Enable “Allow less secure apps” or equivalent setting
   - Use correct SMTP settings for your provider

Problem: “Email configuration not loading”

Symptoms:
- `.env` file exists but settings not applied
- Application ignores email settings

Solutions:

1. **Ensure python-dotenv is installed:**
   ```bash
   pip install python-dotenv
   ```
2. **Load .env file in app.py:**
   ```python
   from dotenv import load_dotenv
   import os

load_dotenv()  # Add this line

app.config['MAIL_SERVER'] = os.getenv('MAIL_SERVER')
app.config['MAIL_PORT'] = int(os.getenv('MAIL_PORT', 587))
```

1. **Restart the application:**
   - Environment variables are loaded at startup

2. **Check .env file location:**
   - Must be in project root directory (same folder as `app.py`)

---

6. UI & Display Issues

Problem: “Charts not showing” / “Blank charts”

Symptoms:
- Dashboard shows empty space where charts should be
- Chart containers visible but no data

Solutions:

1. **Check browser console for errors:**
   - Press F12 → Console tab
   - Look for JavaScript errors

2. **Verify Chart.js is loaded:**
   ```html

```

1. **Check if data is being passed:**
   ```
   python
   # In route
   return render_template('dashboard.html',
       chart_labels=['Mon', 'Tue', 'Wed'],
       chart_data=[10, 20, 30])
   ```

2. **Verify JavaScript chart code:**
   ```
   javascript
   const ctx = document.getElementById('myChart').getContext('2d');
   const myChart = new Chart(ctx, {
       type: 'bar',
       data: {
           labels: {{ chart_labels | tojson }},
           datasets: [{
               data: {{ chart_data | tojson }},
           }]
       }
   });
   ```

3. **Check canvas element exists:**
   ```html

```

Problem: “Bootstrap not working” / “No styling”

Symptoms:
- Pages look unstyled
- Bootstrap components don’t work

Solutions:

1. **Check Bootstrap CDN link:**
   ```html

```

1. **Check internet connection:**
   - CDN requires internet access

2. **Use local Bootstrap (offline solution):**
   - Download Bootstrap and place in `static/` folder

3. **Clear browser cache:**
   - Ctrl+Shift+Delete (or Cmd+Shift+Delete on Mac)

Problem: “Responsive design not working on mobile”

Symptoms:
- Pages don’t adapt to mobile screen size
- Text too small or overflow

Solutions:

1. **Add viewport meta tag:**
   `html`
   `<meta name="viewport" content="width=device-width, initial-scale=1.0">`

2. **Use Bootstrap responsive classes:**
   ```html


   Content

```

1. **Test in browser device mode:**
   - Press F12 → Click device icon → Select mobile device

---

7. API & Route Errors

Problem: "404 Not Found" for API endpoints

Symptoms:

```
404 Not Found: The requested URL was not found on the server
```

Solutions:

1. **Check route is registered:**
   ```python
   @app.route('/api/doctors', methods=['GET'])
   def get_doctors():
       # ...
   ```

2. **Verify correct URL:**
   ```bash
   curl http://127.0.0.1:5000/api/doctors  # Correct
   # NOT: http://127.0.0.1:5000/doctors
   ```

3. **Check blueprints are registered:**
   ```python
   from routes.api_routes import api_bp
   app.register_blueprint(api_bp, url_prefix='/api')
   ```

4. **List all routes:**
   ```python
   # In Python shell
   from app import app
   print(app.url_map)
   ```

Problem: “405 Method Not Allowed”

Symptoms:

```
405 Method Not Allowed: The method is not allowed for the requested URL
```

Solutions:

1. **Add correct methods to route:**
   ```
   python
   @app.route('/api/doctors', methods=['GET', 'POST'])  # Add POST
   def doctors():
       if request.method == 'POST':
           # Handle POST
       return jsonify(doctors)  # Handle GET
   ```

2. **Check request method matches:**
   ```
   bash
   curl -X POST http://127.0.0.1:5000/api/doctors  # POST request
   curl -X GET http://127.0.0.1:5000/api/doctors   # GET request
   ```

Problem: “JSON parsing errors”

Symptoms:

```
werkzeug.exceptions.BadRequest: 400 Bad Request: The browser (or proxy)
sent a request that this server could not understand.
```

Solutions:

1. **Set Content-Type header:**
   ```
   bash
   curl -X POST http://127.0.0.1:5000/api/doctors \
     -H "Content-Type: application/json" \
     -d '{"name":"Dr. Test"}'
   ```

2. **Use request.get_json():**
   ```
   python
   @app.route('/api/doctors', methods=['POST'])
   def add_doctor():
       data = request.get_json()
   ```

```
        if not data:
            return jsonify({'error': 'No JSON data'}), 400
```

3. **Validate JSON structure:**

```
python
    required_fields = ['name', 'email', 'specialization']
    if not all(field in data for field in required_fields):
        return jsonify({'error': 'Missing required fields'}), 400
```

---

8. Performance Issues

Problem: "Application slow to load"

Solutions:

1. **Add pagination to large lists:**

```
python
    page = request.args.get('page', 1, type=int)
    doctors = Doctor.query.paginate(page=page, per_page=20)
```

2. **Use database indexes:**

```
python
    class Doctor(db.Model):
        email = db.Column(db.String(120), unique=True, nullable=False,
index=True)
```

3. **Optimize queries:**

```python
# Bad - Multiple queries (N+1 problem)
for appointment in appointments:
print(appointment.doctor.name)

# Good - One query with join
appointments = Appointment.query.options(db.joinedload(Appointment.doctor)).all()
```

1. **Enable caching (for static data):**

```python
from flask_caching import Cache
cache = Cache(app, config={'CACHE_TYPE': 'simple'})

@cache.cached(timeout=300)
def get_statistics():
```

```
# Heavy computation
```

Problem: “Database getting too large”

Solutions:

1. **Delete old test data:**
   ```python
   from app import app, db, Appointment
   from datetime import datetime, timedelta

with app.app_context():
old_date = datetime.now() - timedelta(days=365)
Appointment.query.filter(Appointment.date < old_date).delete()
db.session.commit()
```

1. **Archive old records:**
   - Move old data to separate archive tables

2. **Use VACUUM to reclaim space:**
   ```bash
   sqlite3 hospital.db "VACUUM;"
   ```

---

9. Port & Network Issues

Problem: “Cannot access application from other devices”

Symptoms:
- Works on localhost but not from other devices on same network

Solutions:

1. **Run on all interfaces:**
   ```python
   if __name__ == '__main__':
       app.run(debug=True, host='0.0.0.0', port=5000)
   ```

2. **Find your local IP:**
   ```bash
   # macOS/Linux
   ifconfig | grep “inet “
   ```

```
# Windows
ipconfig
```

1. **Access using local IP:**
   `http://192.168.1.100:5000  # Replace with your actual IP`

2. **Check firewall settings:**
   - Allow port 5000 in firewall
   - Disable firewall temporarily to test

Problem: "Port blocked by antivirus/firewall"

Solutions:

1. **Add Python to firewall exceptions:**
   - Windows Defender → Allow an app through firewall → Add Python

2. **Use different port:**
   `python`
   `app.run(debug=True, port=8080)  # Try 8080, 8000, 3000`

3. **Temporarily disable antivirus:**
   - Test if application works
   - Add permanent exception if needed

---

10. Submission & Packaging Issues

Problem: "ZIP file too large"

Solutions:

1. **Exclude unnecessary files:**
   `bash`
   `# Don't include:`
   `- venv/ folder`
   `- __pycache__/ folders`
   `- *.pyc files`
   `- .git/ folder`
   `- hospital.db (database file)`
   `- *.log files`

2. **Use create_submission_zip.py script:**
   ```
   bash
     python create_submission_zip.py
   ```

3. **Manually compress correctly:**
   - Only include source code, templates, static files
   - Exclude all generated/cache files

Problem: “ZIP structure incorrect”

Symptoms:
- Validation form shows errors
- Evaluator cannot find main files

Solutions:

1. **Correct structure:**
   ```
   hospital_management_system_2K24XXXXX.zip
     └── hospital_management_system/
         ├── app.py
         ├── requirements.txt
         ├── models/
         ├── routes/
         ├── templates/
         ├── static/
         └── Project_Report.pdf
   ```

2. **Wrong structure (Don’t do this):**
   ```
   hospital_management_system_2K24XXXXX.zip
     ├── app.py  ← Wrong! Files should be inside project folder
     ├── requirements.txt
     └── ...
   ```

3. **Use submission script:**
   ```
   bash
     python create_submission_zip.py
   ```

Problem: “Project doesn’t run after extraction”

Solutions:

1. **Test your submission:**
   ```bash
   ```

```
# Extract your ZIP file
unzip hospital_management_system_2K24XXXXX.zip
cd hospital_management_system

# Create fresh environment
python -m venv venv
source venv/bin/activate

# Install and run
pip install -r requirements.txt
python app.py
```

1. **Include all required files:**
   - [ ] app.py (main file)
   - [ ] requirements.txt
   - [ ] All .py files (models, routes, utils)
   - [ ] templates/ folder
   - [ ] static/ folder
   - [ ] README.md
   - [ ] Project_Report.pdf

2. **Don't include:**
   - [ ] venv/ folder
   - [ ] **pycache**/
   - [ ] hospital.db
   - [ ] .git/
   - [ ] .env file (with passwords)

General Debugging Tips

Enable Debug Mode

```
app.config['DEBUG'] = True
app.run(debug=True)
```

Check Python Logs

```
python app.py 2>&1 | tee app.log
```

Use Python Debugger

```
import pdb; pdb.set_trace()  # Add breakpoint
```

Print Debugging

```
print(f"DEBUG: user = {user}")
print(f"DEBUG: appointments count = {len(appointments)}")
```

Browser Developer Tools

- Press F 1 2
- Check Console tab for JavaScript errors
- Check Network tab for failed requests
- Check Application tab for cookies/session

Database Browser

```
# Install DB Browser for SQLite
# Open hospital.db to inspect data directly
```

Clear Everything and Start Fresh

```
# macOS/Linux
rm -rf venv hospital.db __pycache__
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python app.py

# Windows
rmdir /s venv
del hospital.db
rmdir /s __pycache__
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
python app.py
```

Still Having Issues?

If none of the above solutions work:

1 . **Document the error:**
   - Copy the exact error message

- Note steps to reproduce
- Screenshot if applicable

2. **Check application logs:**
   - Look at terminal output where Flask is running
   - Enable debug mode for detailed errors

3. **Test in isolation:**
   - Create minimal test case
   - Rule out environmental issues

4. **Review code systematically:**
   - Check each component one by one
   - Use print statements to debug

5. **Ask for help:**
   - Post on course forum with error details
   - Include code snippet causing the issue
   - Mention what you've already tried

Prevention Best Practices

1. **Use version control (Git):**
   - Commit working code frequently
   - Easy to revert if something breaks

2. **Test incrementally:**
   - Test each feature as you build it
   - Don't wait until the end to test

3. **Keep backups:**
   ```bash
   cp hospital.db hospital_backup_$(date +%Y%m%d).db
   ```

4. **Use virtual environments:**
   - Isolates project dependencies
   - Prevents conflicts

5. **Document as you go:**
   - Comment complex code
   - Keep notes on issues faced

6. **Follow project structure:**
   - Maintain organized folder structure
   - Use blueprints for routes
   - Separate concerns (models, routes, utils)

Emergency Recovery

If your project is completely broken before deadline:

1. **Stay calm** 🧘
2. **Restore from backup** (if available)
3. **Reset to last working Git commit**
4. **Rebuild from scratch if necessary** (use this guide)
5. **Focus on core features first**
6. **Test thoroughly before submission**

Remember: Most issues have simple solutions. Read error messages carefully, search systematically, and test incrementally. Good luck! 🍀 * *