AI Car Inspector - Sprint Planning Document

Team Name: Dev-Api-Org
Project: AI Car Inspector - Containerized AI Application
Sprint Duration: Tuesday, November 25 - Friday, November 28, 2025
Team Members: Seward (Development), Olefile (Containerization), Dingaan (AWS Deployment)

Team Members & Roles

👷 Seward - Application Development Lead

Responsible for building the core AI-powered Gradio application with Gemini 2.0 Flash integration, implementing the user interface, and ensuring accurate car analysis functionality.

🤽 Olefile - Containerization & CI/CD Lead

Responsible for containerizing the application using Docker, optimizing images, and setting up automated CI/CD pipeline with GitHub Actions for continuous deployment.

☁️ Dingaan - AWS Deployment Lead

Responsible for deploying the containerized application to AWS using Kubernetes/EKS, configuring infrastructure, and completing all project documentation.

Team Member Responsibilities & Deliverables

👷 Seward - Application Development Lead

Tasks:

1. Set up Python 3.11 development environment
2. Implement Gradio UI with minimal, user-friendly design
3. Integrate Gemini 2.0 Flash API (model: `gemini-2.0-flash-exp`, temperature: 0.4)
4. Develop two-stage car analysis logic:
   - Stage 1: Verify if image is a car (yes/no)
   - Stage 2: Extract detailed car information (make, model, year, color, condition)
5. Implement error handling for non-car images and API failures
6. Test application locally with various car images and edge cases
7. Create and validate `requirements.txt`
8. Document API integration and code structure

Deliverables:

- ✅ **app.py** - Fully functional Gradio application
- ✅ **requirements.txt** - Python dependencies
- ✅ **.env.example** - Environment variable template
- ✅ **Local test results** - Screenshots of successful car analysis
- ✅ **Code documentation** - Inline comments and function descriptions

Success Criteria:

- Application runs locally on port 7860
- Correctly identifies car vs non-car images
- Provides accurate car details (make, model, year, etc.)
- Clean, minimal UI with clear user feedback
- Handles errors gracefully

🐋 Olefile - Containerization & CI/CD Lead

Tasks:

1. Create optimized Dockerfile using Python 3.11-slim base image
2. Implement multi-layer build for efficient caching
3. Create .dockerignore to exclude unnecessary files
4. Build and test Docker image locally
5. Create Docker Hub repository and push initial image
6. Configure GitHub Actions CI/CD pipeline (.github/workflows/ci-cd.yaml):
   - Automated testing on push
   - Docker image build with commit SHA tagging
   - Push to Docker Hub registry
7. Set up GitHub repository secrets (DOCKER_USERNAME, DOCKER_PASSWORD)
8. Test automated pipeline end-to-end
9. Document containerization process and pipeline configuration

Deliverables:

- ✅ **Dockerfile** - Optimized container configuration

- ✅ **.dockerignore** - Docker ignore rules
- ✅ **Docker Hub image** - Public repository with tagged images
- ✅ **.github/workflows/ci-cd.yaml** - Complete CI/CD pipeline
- ✅ **Pipeline documentation** - Build process and automation details
- ✅ **Container test results** - Logs showing successful local runs

Success Criteria:

- Docker image builds successfully
- Container runs locally and is accessible on port 7860
- Image successfully pushed to Docker Hub
- CI/CD pipeline triggers automatically on main branch push
- Automated builds complete successfully
- All pipeline stages pass

☁️ **Dingaan - AWS Deployment Lead**

Tasks:

1. Set up AWS EKS cluster or configure existing cluster access
2. Update Kubernetes manifests with correct Docker Hub image URL
3. Create Kubernetes Secret for Gemini API key:

```bash
kubectl create secret generic car-inspector-secret \
  --from-literal=GEMINI_API_KEY=<api_key>
```

4. Deploy application to AWS:
   - Apply deployment.yaml (2 pod replicas with resource limits)
   - Apply service.yaml (LoadBalancer configuration)
5. Configure AWS security groups and networking for external access
6. Verify deployment health and pod status
7. Test application accessibility via LoadBalancer URL
8. Set up basic monitoring and logging
9. Capture deployment screenshots and logs

10. Complete capstone_reflection.md with architecture diagrams
11. Prepare presentation materials
12. Finalize all submission documentation

Deliverables:

- ✅ **kubernetes/deployment.yaml** - K8s deployment configuration
- ✅ **kubernetes/service.yaml** - K8s service configuration (LoadBalancer)
- ✅ **kubernetes/secret.yaml** - Secret template (no real keys)
- ✅ **AWS deployment evidence**:
  - Screenshots of running pods (`kubectl get pods`)
  - Service details with external URL (`kubectl get services`)
  - Application logs (`kubectl logs`)
- ✅ **Live application URL** - Accessible application endpoint
- ✅ **README.md** - Complete project documentation
- ✅ **capstone_reflection.md** - Detailed project reflection with:
  - Architecture diagrams
  - Challenges and solutions
  - Key learnings
  - Testing results
- ✅ **Presentation materials** - Supporting documentation for organizational presentation

Success Criteria:

- Application successfully deployed to AWS EKS
- Pods running and healthy
- External LoadBalancer URL accessible from browser
- Application responds correctly to car image uploads
- Resource limits properly configured
- Complete documentation ready for submission
- Presentation materials prepared

Sprint Timeline

Sprint Duration: Tuesday, November 25 - Friday, November 28, 2025

Daily Sync Meetings: 13:00 PM (after lunch)

Status Updates: Shared after completing key deliverables

Communication Protocol: Team members reported progress and blockers during daily syncs, with ad-hoc updates as major milestones were achieved throughout the sprint.

Integration Points & Dependencies

Seward → Olefile

- **Handoff**: Completed `app.py` and `requirements.txt`
- **Timing**: After local testing confirms application works
- **Communication**: Share .env.example with API key format requirements

Olefile → Dingaan

- **Handoff**: Docker Hub image URL and tag
- **Timing**: After successful Docker push and CI/CD pipeline test
- **Communication**: Provide exact image path for deployment.yaml

All → Dingaan

- **Handoff**: All code files, test results, and process documentation
- **Timing**: Throughout sprint for reflection document
- **Communication**: Share screenshots, logs, and learnings for documentation

---

Quality Checklist

Seward - Development

- [ ] Application runs without errors locally
- [ ] All dependencies listed in requirements.txt
- [ ] Environment variables documented
- [ ] Car detection accuracy tested (car vs non-car)
- [ ] Error handling implemented
- [ ] Code commented and clean

Olefile - Containerization

- [ ] Dockerfile builds successfully
- [ ] Container runs locally

- [ ] Image pushed to Docker Hub
- [ ] CI/CD pipeline passes all stages
- [ ] GitHub secrets configured
- [ ] Documentation complete

Dingaan - Deployment

- [ ] Pods running and healthy on AWS
- [ ] LoadBalancer accessible externally
- [ ] Application responds correctly
- [ ] Screenshots captured
- [ ] Documentation complete
- [ ] Presentation materials ready
- [ ] All files in GitHub repository

Final Submission Checklist

GitHub Repository Must Contain:

- [ ] app.py
- [ ] requirements.txt
- [ ] Dockerfile
- [ ] .dockerignore
- [ ] .env.example
- [ ] kubernetes/deployment.yaml
- [ ] kubernetes/service.yaml
- [ ] kubernetes/secret.yaml (template only)
- [ ] .github/workflows/ci-cd.yaml
- [ ] README.md
- [ ] capstone_reflection.md

Additional Deliverables:

- [ ] Docker Hub image link
- [ ] AWS LoadBalancer URL
- [ ] Screenshots of running application
- [ ] Deployment evidence (kubectl outputs)
- [ ] CI/CD pipeline success logs
- [ ] Presentation materials

Success Metrics

Application Performance:

- Fast response times
- Accurate car detection and classification
- Stable and reliable operation

DevOps Excellence:

- Efficient build and deployment processes
- Optimized container images
- Successful automated CI/CD pipeline
- Zero failed deployments

Documentation Quality:

- Complete architecture diagrams
- Clear setup instructions
- Comprehensive project reflection
- Professional presentation materials

Sprint Goal: Deliver a fully functional, containerized, AI-powered car inspector application deployed on AWS with complete documentation by November 28, 2025 COB.

Let's build something amazing! 🚀