








Project Title: AutoSpec.AI

Smart Document-to-System Requirements Analyzer

Project Summary

AutoSpec.AI is a serverless application that enables users to email or upload documents (PDF, DOCX, TXT) and receive structured, AI-generated system requirements using a Bedrock-powered prompt that mimics the mindset of a skilled systems analyst. The output includes functional requirements, non-functional requirements, stakeholder roles, and technical transparency in both narrative and tabular format.

Hackathon Requirement Compliance

Requirement	Met?	Notes
Uses AWS Lambda		Lambda handles all processing, triggering, and orchestration
Serverless architecture		All components (S3, SES, Bedrock, DynamoDB, etc.) are serverless
Integrates AWS services		Bedrock, S3, SES, EventBridge, DynamoDB, API Gateway
Code and README hosted in public GitHub repo		Documented with architecture and deploy instructions
Explanation of Lambda use		Included in README and video
Built during the hackathon		New project with phased delivery plan
3-minute demo video		Will show doc upload, Bedrock processing, and response via email/Slack

System Design Overview

Core Functionalities:

1. Receive document input (via email or API upload).
 2. Store document in S3.
 3. Trigger Lambda function on new upload.
 4. Parse document (PDF, DOCX, TXT).
 5. Call Amazon Bedrock with a well-crafted prompt.
 6. Format AI response into structured system requirements (Markdown, JSON, optionally PDF).
 7. Send result back via email (SES) or Slack webhook.
 8. Save input/output history in DynamoDB with metadata for version tracking.
-

Architecture Diagram

[Email or API Upload]



[Amazon SES / API Gateway]



[S3 Bucket] ←————→ [CloudWatch Logs]

↓ (Object Created Event)

[Lambda Function #1: DocumentProcessor]



[Amazon Bedrock → Claude 3.7]



[Lambda Function #2: Formatter]



[DynamoDB: store history + metadata]



[SES / Slack → Response Sent to User]

Development Plan

Week 1: Core Functionality MVP

Day 1–2: Environment Setup

- Initialize GitHub repo with CDK or SAM template.
- Set up S3, SES (verified email), API Gateway (optional), IAM roles.

Day 3–4: Lambda: Document Ingestion

- SES inbound rule → Lambda trigger.
- S3 upload handler Lambda to log metadata and route to processing.

Day 5–6: Lambda: AI Processing

- Format document content.
- Send to Amazon Bedrock with system analysis prompt.
- Capture Bedrock response (raw and structured).

Day 7: Lambda: Output Generation + Return

- Structure output: Markdown + JSON.
- Send response via SES.
- Store record in DynamoDB with original file + response hash.

Week 2: Polish and Enhance

Day 8–9: Slack Integration (optional path)

- Add support for receiving/responding via Slack slash command or webhook.

Day 10: Output Formatting

- Convert response to downloadable PDF (using Lambda+Layer or prebuilt utility).
- Add summary chart/table formatting.

Day 11: API Gateway (alternate input)

- Support RESTful upload of file → same Lambda workflow.

Day 12: Logging + Monitoring

- Add detailed logging (CloudWatch) and metrics for throughput/error rate.

Day 13: README + Deployment Script

- Add instructions for self-deploy.
- Include architecture diagram and Lambda flow explanation.

Day 14: Record 3-minute demo video

- Show full flow: email in → Bedrock → structured AI response.

Optional Future Enhancements (If Time Permits)

Feature	Description
Web UI	Simple UI to upload document and retrieve response
OAuth	Email verification + upload access control
Document Diff	Compare new vs old requirements (versioning)
Multi-model AI	Offer comparison between Claude, Titan, and Jurassic responses
Fine-tuned prompts	Select from different requirement formats (Agile user stories, Use Cases, etc.)
Batch input	Accept multiple documents in zip and return consolidated results
Feedback loop	Allow users to rate output quality and fine-tune prompts accordingly

Prompt for Bedrock

Review this document with the mindset of a skilled systems analyst and break down the contents into an organized list of system requirements that meet these criteria:

1. Comprehensive and Clear Structure
2. Depth and Clarity of Functional Requirements
3. Role-Specific Responsibilities

4. Technical Transparency
5. Non-Functional Requirements Are Not Forgotten
6. Cross-Referencing and Version Control
7. Balanced Use of Narrative and Tabular Format

Return your output in structured Markdown with headings for each requirement category.

GitHub Repository Structure

/autospec-ai/

├── /lambdas/

| ├── ingest/

| ├── process/

| └── format/

├── /docs/

| └── architecture.md

├── /infra/

| └── cdk/ or sam/

├── README.md

├── demo.mp4

└── deployment_instructions.md