Acronym/Abbreviation	Full Term
ALPR	Automatic License Plate Recognition
FR	Functional Requirement
QA	Quality Attribute

CST	Constraint
TE	Technical Experiment
ADR	Architecture Decision Record

Previously on M1

No	Interview Comment(Feedback)	Decision & Updates
1	The plan is not clear. Separation of level 1, 2, 3 is not intuitive. A Gantt chart notation and more coarse grained activities could be easier to create and maintain.	We updated the timeline for better visualization.
2	Why separate requirements of laptop application and server application? Splitting quality attributes rqmts between client and server is problematic. Identify system level quality attributes in addition to individual components.	We combined those functional requirements divided into a client and a server.
3	QA about performance on the server is not clear.	We reviewed and updated all the quality attributes one by one together.
4	A data access layer on top of Berkeley DB, Maria DB, and Mongo DB seems like overdesigning. These are 3 different types of DB (key-value, RDBMS, document). Do we need that level of abstraction in terms of persistence operations?	We were supposed to choose one of those 3 DBs through technical experiment, **not** abstract. Anyway, we have been going through trial and error about using DB for partial matches so we have reviewed 3 DBs in result.
5	Some experiments are planned.	We made progress on those experiments.
6	One experiment is focusing on technology selection rather than being able to fulfill requirements and constraints.	As you said, we're not familiar with technologies for a server and NoSQL. so we try to investigate what skill is best for our project. but when choosing technologies, we consider requirements and constraints first.
7	Using RUP is a surprise	Although it is not easy to follow RUP exactly to our short term project, it is helpful for us to study something new.
8	If the design is to be done as presented, then start constructing technical experiments as soon as possible to see if it will even work	We already had a working (initial version of) system based on technical experiments.
9	Quality attributes are untethered somewhat, and some values come up with assumptions.	We eliminate vague values and designate specific values from the project description. and we piece together requirements concerning the same problem.
10	The design is web based which is a little complicated. There seems to be agreement that there is technical risk.	We have verified that the system works through to reduce technical risk.
11	Need to run an experiment soon to make sure React and open ALPR will work fine.	Same as above.

Previously on M2

No	Interview Comment(Feedback)	Decision & Updates
1	Can't tell what is done and what isn't in their gantt chart.	All the items on the gantt chart are done.
2	Deployment view is missing important information.	We revise the deployment view containing detailed importation.
3	The views are somewhat disconnected - the diagram is not co-located with the catalog.	We rewrite QAs, ADR and relevant diagrams.and we think it would make sense.
4	Good use of the ADR. The view must contain the documentation for that view, which includes the diagram as well as the written text. Review what a deployment view is and revise the deployment view in the architecture. The ADR should be reflected in the architecture. Ordering and documentation cohesion is the major issue here.	We revise the deployment view containing detailed importation. We rewrite QAs, ADR and relevant diagrams. In Particular, we rewrite the rationale of ADR to show how it is reflected in the architecture.
5	The React web UI experiment is still going on.	We’ve finished the experiment and development of React web UI.
6	Architecture views do not follow a template. The views are basically a sequence of diagrams. They are all grouped under a high level section called “Primary presentation”. Then, there’s a separate section “Element Catalog”. Each view should have a primary presentation, element catalog, etc.	We reorganized the template. and rearranged all the architecture design sections. so all the contents in the architecture section follow the same template.
7	The deployment view shows the client laptop and server, but there’s no information about these elements. What are relevant properties (memory, OS, bandwidth, etc.)?	We revise the deployment view containing detailed importation. and It has all of the relevant information needed.
8	Each ADR should be a separate document. The architecture views could have links to the ADRs (and vice-versa).	We separate the tangled ADR. Now each ADR has its own unique subject.
9	Overall Client-Server view shows “License Plate DB”. However, I couldn’t find any information about the type of DB. Is it Maria DB? Solr? MongoDB? The choice of DB has a direct impact in achieving some of the quality requirements. (And functional rqmts, as in storing duplicate license plates and handling partial matches.)	We specified the specific type of DB. Actually we chose MongoDB as our database.
10	How does the server verify it’s an authenticated user that sent the plate query request? A sequence diagram showing authentication and use of JWT tokens for the purpose would be useful. BTW, no behavior diagram was created.	We modify the sequence diagram to show how the server verifies the user's token.
11	Nice that you included a variability guide.	Thanks.

Project Plan

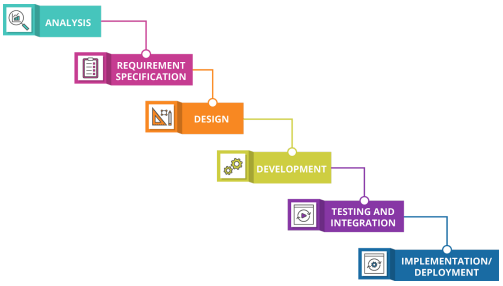
At a glance

The following 7 questions arose when we first found out about the project:

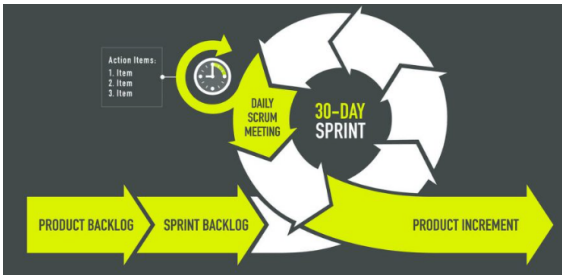
1. Just six weeks to develop the ALPR system?
2. What if the system is unavailable during a critical period?
3. What if a hacker breaks into the network?
4. What if a car is not in the view before the result comes out?
5. What if there are too many cars so the system cannot respond in time?
6. What if the system gives a wrong answer?
7. What if the CPU performance becomes low?

Considering software development process

We have discussed which development process would be best for us to complete our mission within a short period of time. First, we decided to exclude the *Waterfall* model (or a linear-sequential life cycle model) because we do not have enough time to determine all the requirements and quality attributes at the beginning.

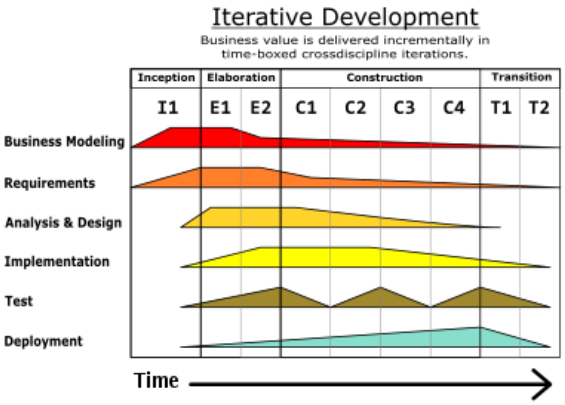


vs.



On the other hand, *Scrum*(or *Agile* methodology) seems to be good as it is flexible to change, guarantees high quality/fast delivery and continuous improvement. However, we do not have enough time to use *Scrum* since it usually works with 2-week-sprint.

Finally, we decided to use *Rational Unified Process*, which is still iterative and incremental, architecture-centric, risk-focused, and widely used in web applications, similar to our project.



Taking *Rational Unified Process* into consideration, we made the following schedule.

Role divisions

- Sangdong Ahn : Server (DB)
- WooYong Chung : Server (WAS)
- Heewan Park : Server (DB)
- Ruben Choi : Client (UI, Connection)
- Jinok Kim : Client (Tesseract)
- Sanghwa Yu : Client (Https)

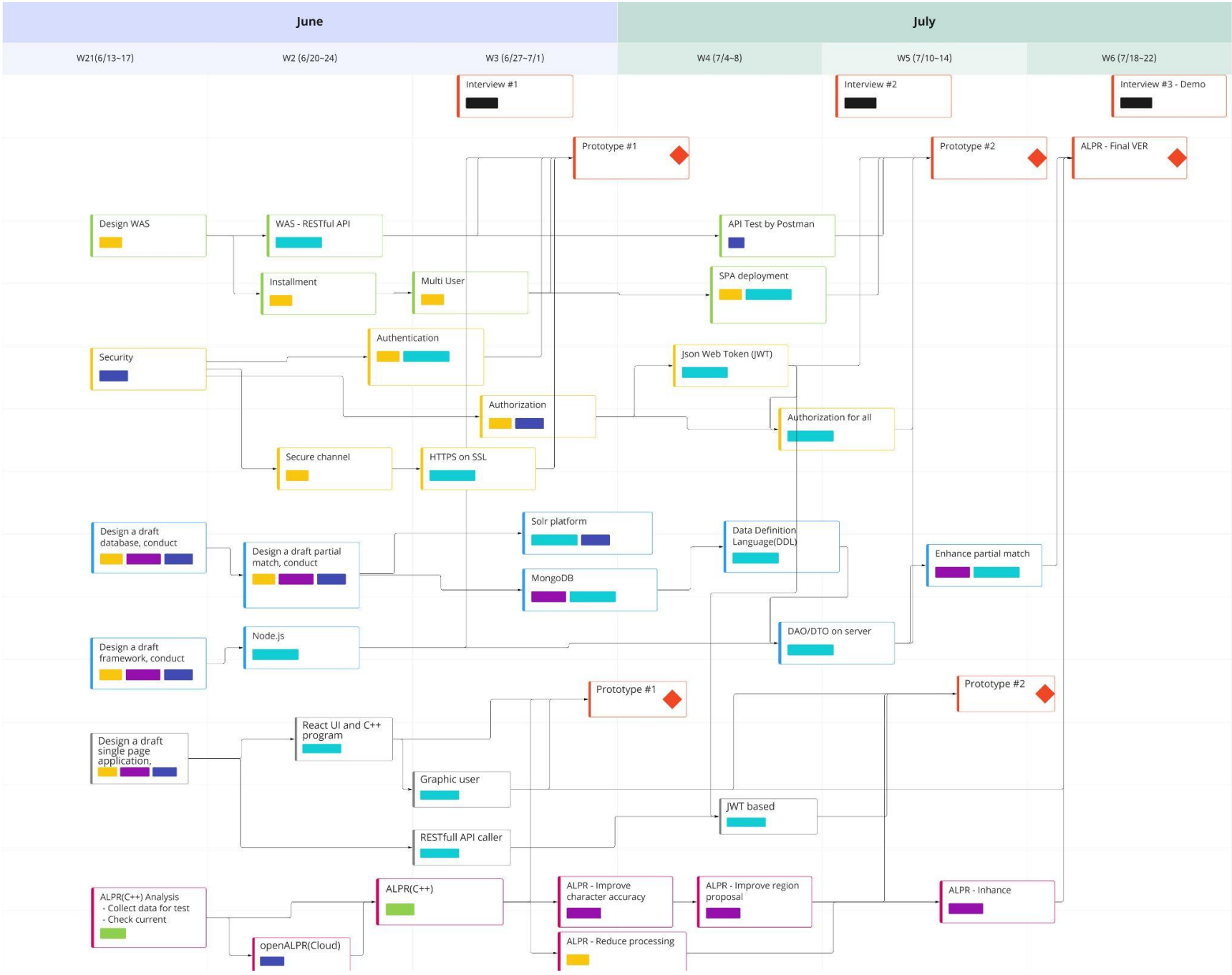
Schedule

The detailed plan written in the Gantt chart can be found at the URL below.
https://miro.com/app/board/uXjVOtKoTGk=/
It is also attached to the next page.

Goals with Quality Attributes

Analyzing the functional requirements (which are following the schedule section below), it became clear what kind of system we wanted to build. That is, a system satisfying the 7 questions, where each topic is mapped to a specific quality attribute:

1. Just six weeks to develop the ALPR system? **Modifiability**
2. What if the system is unavailable during a critical period? **Availability**
3. What if a hacker breaks into the network? **Security**
4. What if a car is not in the view before the result comes out? **Performance**
5. What if there are too many cars so the system cannot respond in time? **Performance**
6. What if the system gives a wrong answer? **Accuracy**
7. What if the CPU performance becomes low? **Usability**


June
July
W21(6/13~17)
W2 (6/20~24)
W3 (6/27~7/1)
W4 (7/4~8)
W5 (7/10~14)
W6 (7/18~22)
Interview #1
Interview #2
Interview #3 - Demo
Prototype #1
Prototype #2
ALPR - Final VER
Design WAS
WAS - RESTful API
API Test by Postman
Installment
Multi User
SPA deployment
Authentication
Security
Json Web Token (JWT)
Authorization
Authorization for all
Secure channel
HTTPS on SSL
Design a draft database, conduct
Design a draft partial match, conduct
Solr platform
MongoDB
Data Definition Language(DDL)
Enhance partial match
Design a draft framework, conduct
Node.js
DAO/DTO on server
Prototype #1
Prototype #2
Design a draft single page application,
React UI and C++ program
Graphic user
JWT based
RESTfull API caller
ALPR(C++) Analysis
- Collect data for test
- Check current
ALPR(C++)
ALPR - Improve character accuracy
ALPR - Improve region proposal
ALPR - Inhance
openALPR(Cloud)
ALPR - Reduce processing

Architectural Drivers

miro

Functional Requirements

ID	Functional Requirement	Description
FR-01	Select File	The system should allow the user to select a playback file as input.
FR-02	Dynamic Configuration	The system should support configurable values via a configuration file. - Minimum confidence threshold to support a partial match - Frame skip rates on laptop application preview - expiration time : 1/2/3/4 hours.
FR-03	Show Playback	The system should display playback video in the playback view area.
FR-04	Show Performance	The system should display a playback frame per second / average time per frame / jitter / frame number in the playback view area.
FR-05	Show Identifying Result	The system should display the recognized plate image, recognized plate number, vehicle maker, vehicle model, vehicle color in the playback view area.
FR-06	Alert Information	The system should display reason, owner name, address, vehicle make, model, color, isolated plate image, recognized license plate number in the alert area.
FR-07	Alert Network Error	The system should alert any network communication error or failures.
FR-08	Support Multi-user	The system should support multi-user support.
FR-09	Identify US License Plate	The system should identify US license plates from images in the playback file.
FR-10	Measure Performance	The system should measure a playback frame per second / average time per frame / jitter / frame number.
FR-11	Store Car Information	The system should store the following contents. - license plate number - Status : Owner Wanted / Stolen / Unpaid Fines – Tow / No Wants/Warrants - Registration Expiration date - Owner Name - Owner Date of Birth - Owner Street Address / Location - Owner City, State and Zip Code - Vehicle Year of Manufacture - Vehicle Make - Vehicle Model - Vehicle Color
FR-12	API Monitoring Server Performance	The system should track the following metrics. - the average number of queries per second for each user - the average number of queries per second for all users

		- the number of partial matches for each user - the number of no matches for each user - the number of partial matches for all users - the number of no matches for all users
FR-13	Detection	ALPR shall find potential license plate regions for further processing.
FR-14	Character Analysis	ALPR shall find character-sized "blobs" in the plate region
FR-15	Plate Edges	ALPR shall the edges/shape of the license plate
FR-16	Deskew	ALPR shall transform the perspective to a straight-on camera view based on the typical license plate sizes
FR-17	Character Segmentation	ALPR shall isolate characters so that they can be processed individually
FR-18	Optical Character Recognition	ALPR shall analyze each character image and provides possible letters/confidences
FR-19	Syntactical/Geometrical analysis	ALPR shall check characters and positions against state-specific formats
FR-20	Accuracy	ALPR shall provide with the evaluated accuracy
FR-21	Manage work request	API calls should be minimized considering the load on the server.
FR-22	Partial Match	The system should return the best match license plate if there is not an exact match that includes a configurable minimum confidence threshold to support a partial match.
~~FR-23~~	~~maximize recognition accuracy~~	~~ALPR shall maximize recognition accuracy~~
FR-24	Security	Only authorized users can use the server's API to query car information based on plate numbers.
FR-25	Security	The server should support user authentication.

Quality Attributes

ID	Attribute	Description
QA-01	Modifiability	The UI developer should be able to change and test the user interface within 3 hours.
QA-02	Availability	The client should detect and alert the user about network connectivity issues with the backend system within 3 seconds and automatically try to reconnect for every 1 second.
QA-03	Security	The system should detect an illegitimate access and notify the administrator immediately when an unauthorized user attempts to access the system API.
QA-04	Performance	Plate number query on the server should be done within 1 second, when DB has 25,000,000 records for car detection.

QA-05	Performance	The system's ALPR detection for 25 frames of 640x480 video footage should be done within 1 second.
QA-06	Accuracy	ALPR shall maximize recognition accuracy
QA-07	Usability	The user can change the rate of the sending ALPR preview image fps and the preview is displayed with the set rate once the user adjusted.

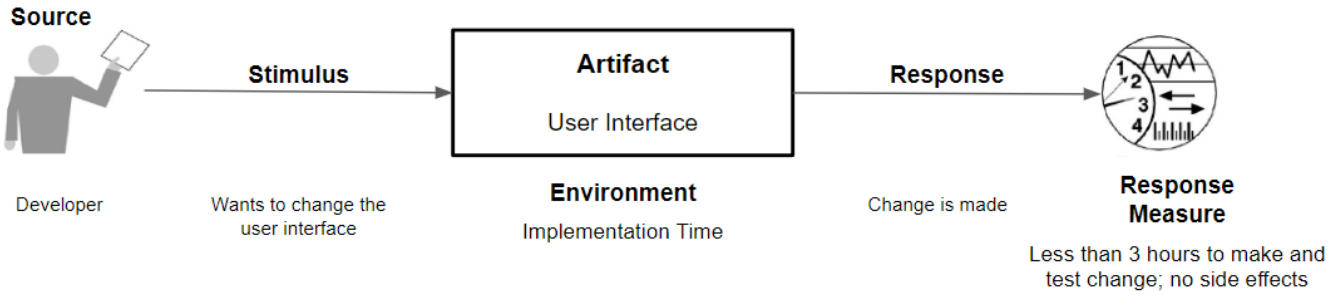
Constraints

ID	Type	Description
CST-01	Technical	The system must run on Windows 10 laptop
CST-02	Business	The system supports vehicle DB in Pennsylvania state of the US only
CST-03	Business	Valid user list(police) is set on the User table
CST-04	Technical	No DNS available(No Official Root Certificate) at WeWork Office.
CST-05	Business	The project ends in 5 Weeks.
CST-06	Technical	A user must proceed to the unsafe certificate prior to use the system. (See Experiment 5: Install X.509 Certificate for more detail.)
CST-07	Technical	The connection between server and client should be secured.
CST-08	Business	The driver records at a low speed (under 30 km/h) to examine cars.

Quality Attribute Scenario

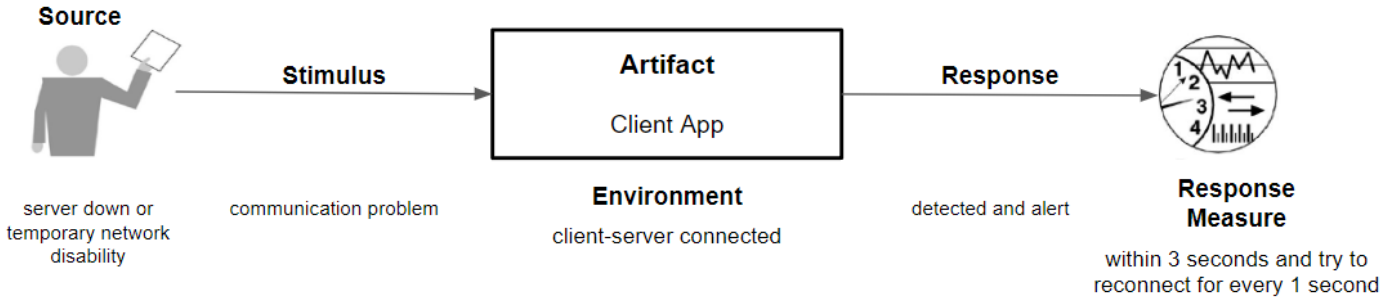
1. QA-01

Quality Attribute	Description
Scenario Description	The UI developer wants to change the user interface.
Stimulus	Wants to change the user interface
Stimulus source	Developer
Artifact	User Interface
Environment	Implementation time
Response	Change is made
Response measure	Less than 3 hours to make the test change; no side effect



2. QA-02

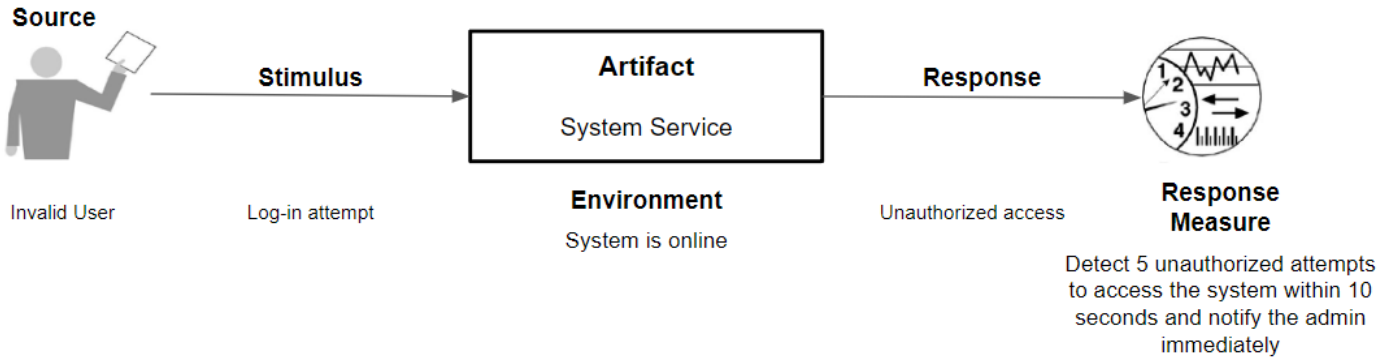
Quality Attribute	Description
Scenario Description	The client is connected to the server. If there is a communication problem, it should detect and alert the user within 3 seconds and try to reconnect for every 1 second.
Stimulus	communication problem
Stimulus source	server down or temporary network disability
Artifact	client app
Environment	client-server connected
Response	detected and alert
Response measure	within 3 seconds and try to reconnect for every 1 second



3. QA-03

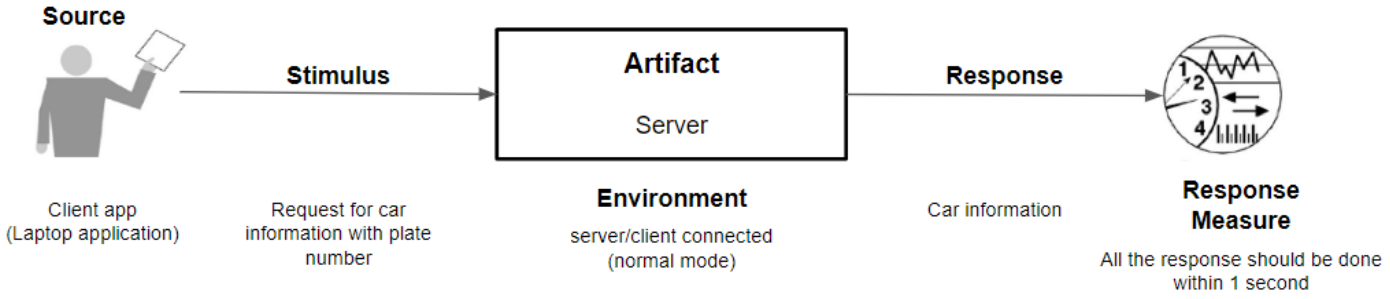
Quality Attribute	Description
Scenario Description	The server should support user authentication.
Stimulus	Log-in attempt
Stimulus source	Invalid User
Artifact	System service
Environment	System is online

Response	Unauthorized access
Response measure	Detect 5 unauthorized attempts to access the system within 10 seconds and notify the admin immediately



4. QA-04

Quality Attribute	Description
Scenario Description	Plate number query on the server should be done within 1 second, when DB has 25,000,000 records for car detection.
Stimulus	Request for car information with plate number
Stimulus source	Client app(Laptop application)
Artifact	Server
Environment	server/client connected (normal mode)
Response	Car information
Response measure	All the response should be done within 1 second



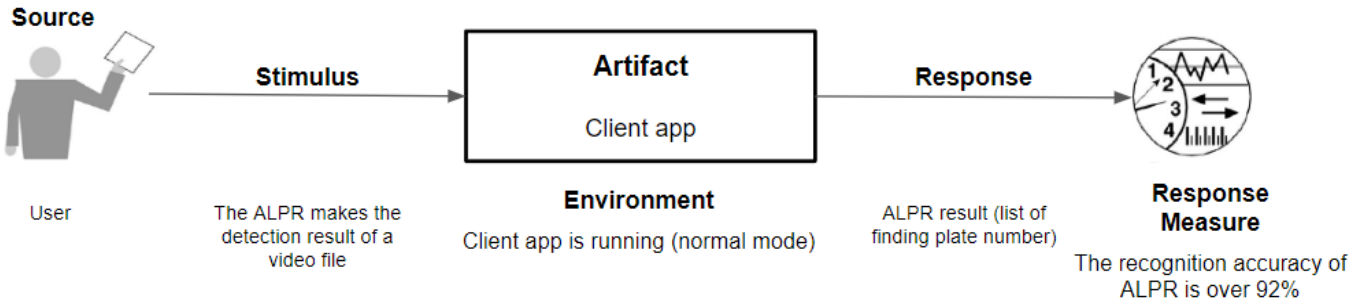
5. QA-05

Quality Attribute	Description
Scenario Description	The system's ALPR detection for 25 frames of 640x480 video footage should be done within 1 second.

Stimulus	User selects a playback file (640x480 video footage) and run ALPR
Stimulus source	User
Artifact	Client app(Laptop application)
Environment	Client app is running (normal mode)
Response	ALPR result (list of finding plate number)
Response measure	Even at the peak moments ALPR detection for 25 frames of 640x480 video footage is done within 1 second.

6. QA-06

Quality Attribute	**Description**
Scenario Description	The recognition accuracy[1] of ALPR shall be over 92%[2]
Stimulus	The ALPR makes the detection result of a video file.
Stimulus source	User
Artifact	Client app(Laptop application)
Environment	Client app is running (normal mode)
Response	ALPR result (list of finding plate number)
Response measure	The recognition accuracy of ALPR is over 92%

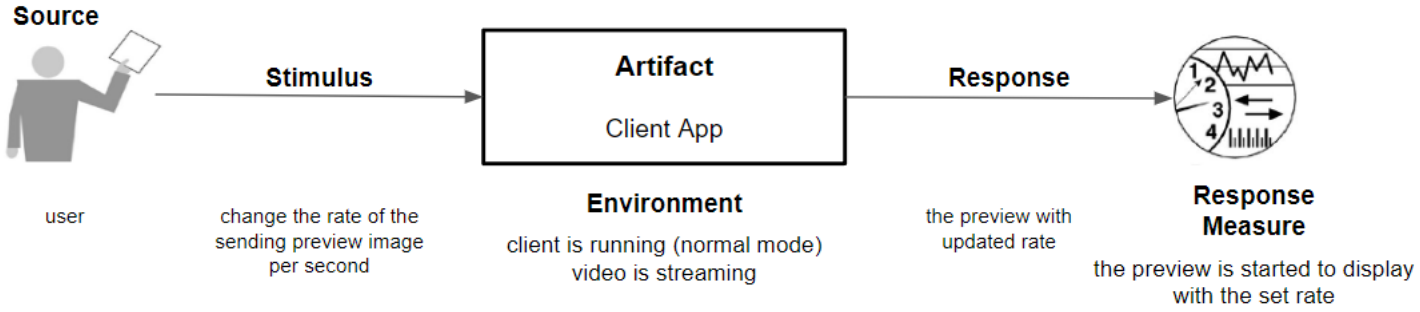


7. QA-07

Quality Attribute	**Description**
Scenario Description	A user can adjust the rate of the sending ALPR preview image (per second) according to their CPU performance.
Stimulus	Change the rate of the sending preview image per second

[1] Defined accuracy : The number of correctly detected license plate numbers among existing license plate numbers.
[2] Reason for 92% :The accuracy of given detection logic was 90.12% and our goal is to improve by at least over 2 percent. We recognized in many papers that it's hard work to improve the accuracy by at least over 1 percent in the deep learning field.

Stimulus source	User
Artifact	Client app
Environment	Client app is running (normal mode) and video is streaming from ALPR
Response	The preview with updated rate
Response measure	The preview is started to display with the set rate



Risk Assessment/Planned Experiments

Risk Assessment

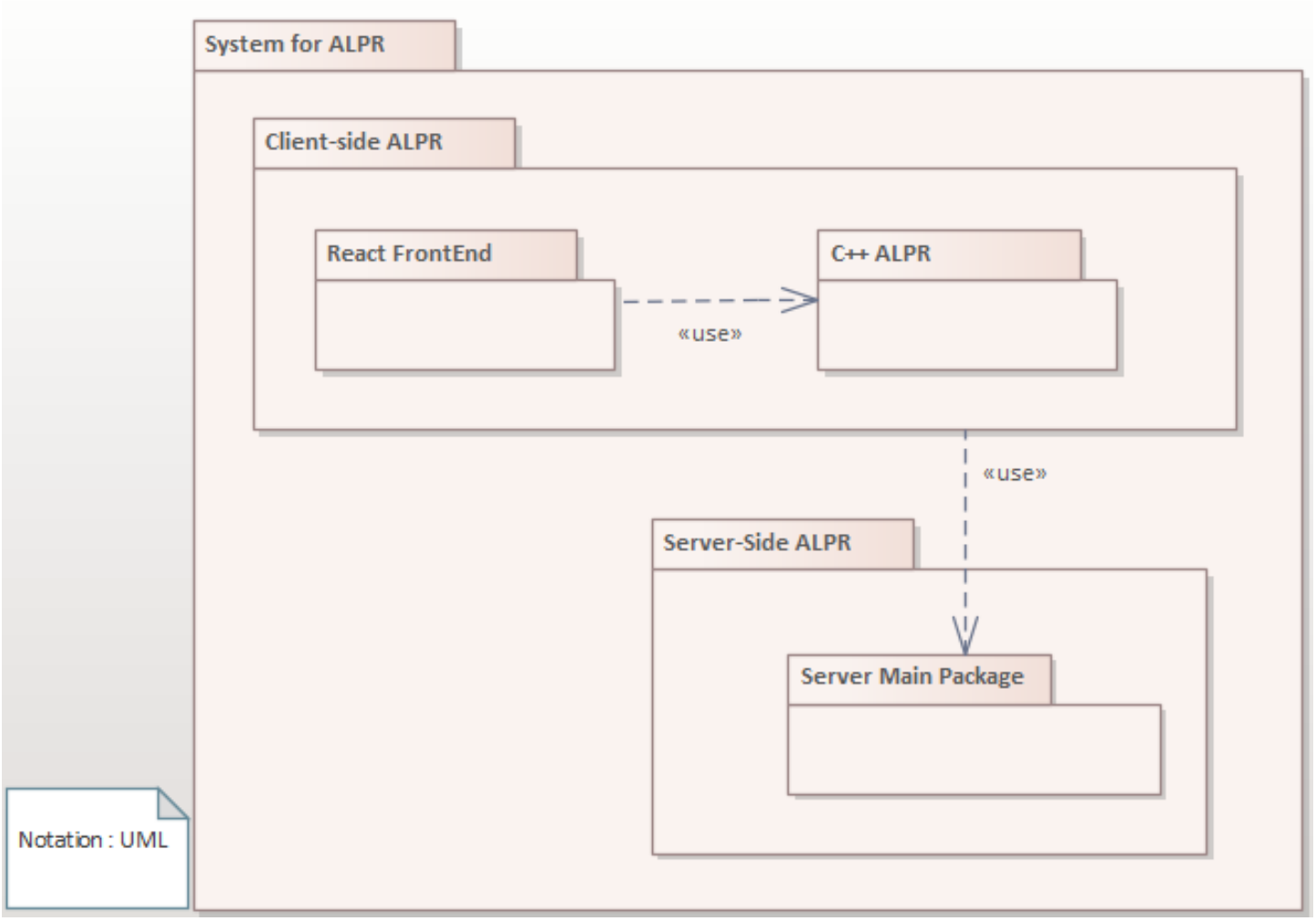
ID	TYPE.	Description	Technical Difficulty	Business Importance	Is Risk	Need Experiment
QA-01	Modifiability	The UI developer should be able to change and test the user interface within 3 hours.	H	H	N	Y TE#3
QA-02	Availability	The client should detect and alert the user about network connectivity issues with the backend system within 3 seconds and automatically try to reconnect for every 1 second.	M	M	N	N
QA-03	Security	The system should detect an illegitimate access and notify the administrator immediately when an unauthorized user attempts to access the system API.	L	M	N	N
QA-04	Performance	The system should respond to plate number detection requests within 1 second when DB has 25,000,000 records for car detection.	M	H	Y	Y TE#1 TE#2 TE#6
QA-05	Performance	The system should perform ALPR function in real -time while maintaining a frame rate of at least 25fps.	H	H	Y	Y TE#3
QA-06	Accuracy	ALPR shall maximize recognition accuracy	M	H	Y	Y TE#7

QA-07	Usability	The user can change the rate of the sending ALPR preview image fps and the preview is displayed with the set rate once the user adjusted.	M	H	N	Y TE#4

Architectural Design

Static perspective - Module Views

Top-level ALPR module view



Element Catalog

Element	Responsibility
React Front-end	The front-end view for user interaction
C++ ALPR	The ALPR engine detecting plate numbers from the given video stream
Server Main Package	The server with the license plate database providing authorized REST API

Element Interfaces

N/A

Element Behavior

N/A

Variability Guide

N/A

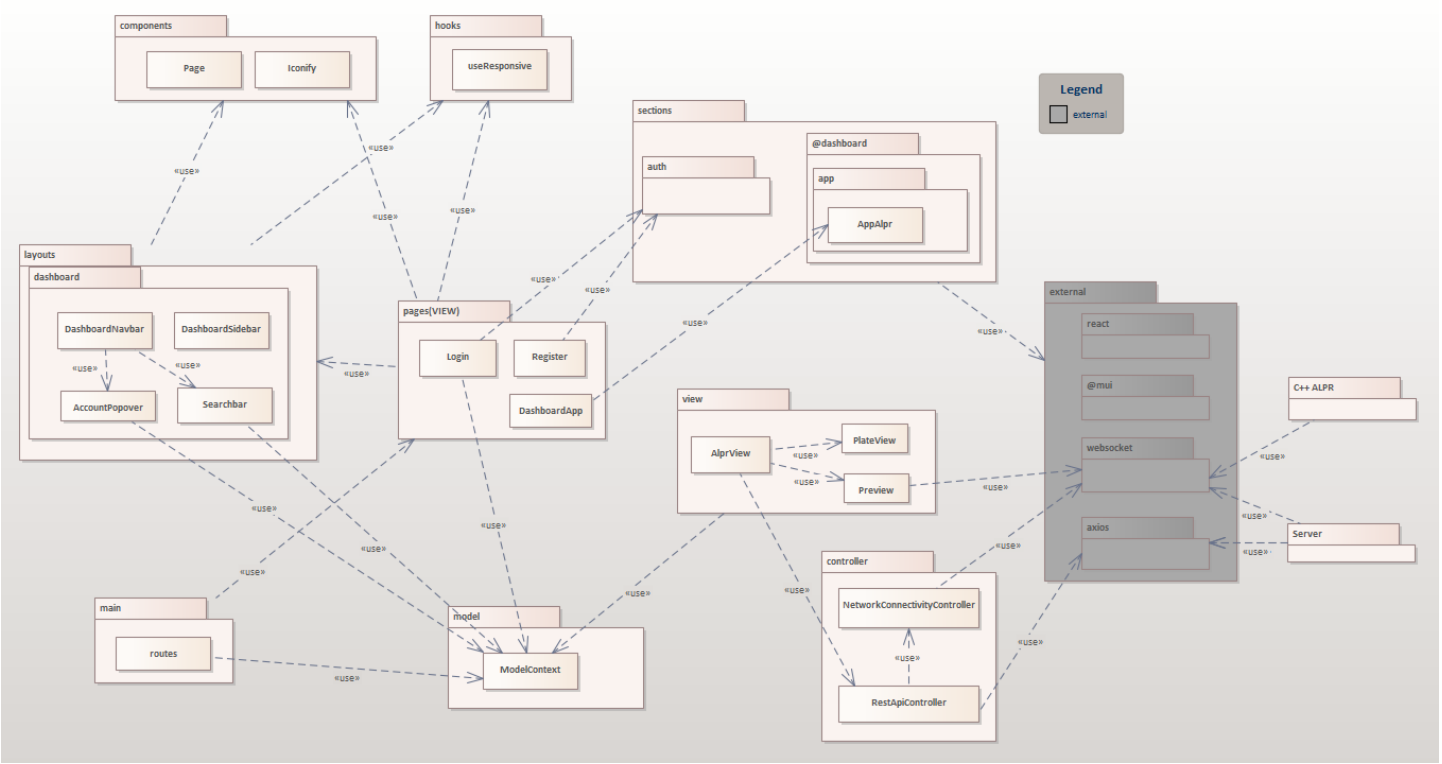
Rationale

1. ADR 2: Use React Front-end
2. ADR 5: Use C++ openALPR as the license plate recognition engine
3. ADR 7: Use Node.js Express Server

Related views

1. Refinement React Front-End module uses view
2. Refinement C++ ALPR module uses view
3. Refinement Server-side module uses view

Refinement React Front-End module uses view



Element Catalog

Element	Responsibility
view/pages	provides a role as view in MVC
controller	provides a role as controller in MVC
model	provides a role as model in MVC
Preview	communicating with C++ ALPR engine via websocket, provides a UI menu to select a playback file (FR-01) provides a UI menu to select the flow control parameter of the preview(FR-02) provides a UI preview (FR-03, FR-04) provides a UI popup to show the identifying result (FR-05)
RestApiController	provides a communication with the server via REST API
NetworkConnectivity Controller	provides a UI alert on the identifying result or the network errors (FR-06, FR-07, QA-02)
route	provides the entry point of the system

Element Interfaces

N/A

Element Behavior

N/A

Variability Guide

N/A

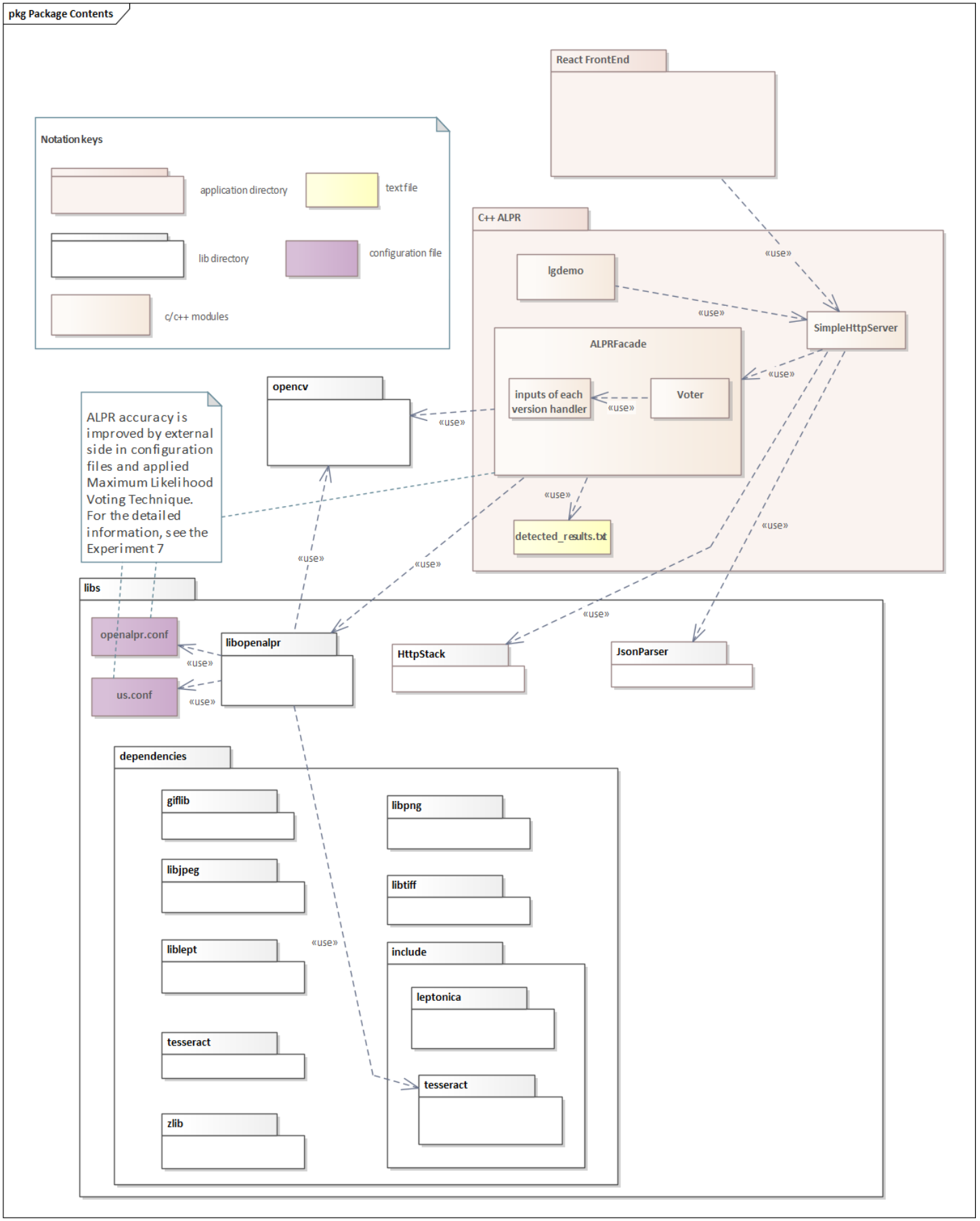
Rationale

1. ADR 1: Use Model-View-Controller(MVC) pattern

Related views

1. Top-level ALPR module uses view

Refinement C++ ALPR module uses view



Element Catalog

Element	Property	Responsibility
LGDemo	main	C++ Main function to run OpenALPR
ALPRFacade DashboardApp	detect	start ALPR detection with filepath, frame skip rates with callback Record whole detected results in text data file(detected_results.txt) and accuracy is calculated after execution is done internally.(FR-20)
	stopDetect	stop working ALPR processing.
	-	The accuracy is improved by external configuration data files(openalpr.conf and us.conf) and applied Maximum Likelihood Voting Technique(QA-06)
SimpleHttpServer	WebSocketData	websocket interface to communicate with React frontend.
	SendWebSocket Packet	Callback function to send ALPR results (preview, plate number and so forth) to React frontend.
libopenalpr	-	Find potential license plate regions for further processing(FR-13) Find character-sized “blobs” in the pate region(FR-14) Find the edges/shape of the license plate(FR-15) Transform the perspective to a straight-on camera view based on the typical license plate sizes(FR-16) Isolate characters so that they can be processed individually(FR-17) Check characters and positions against state-specific formats(FR-19) The accuracy is improved by external configuration data files(openalpr.conf and us.conf) and applied Maximum Likelihood Voting Technique(QA-06) When the detected result is different compared with the previous result, ALPR sends the result to the server. As accuracy is improved by external configuration data files, the amount of results sent to the server is reduced than before. Thus, it makes minimized calls.(FR-21)
tesseract	-	Analyze each character image and provides possible letters/confidences(FR-18)

Element Interfaces

N/A

Element Behavior

N/A

Variability Guide

N/A

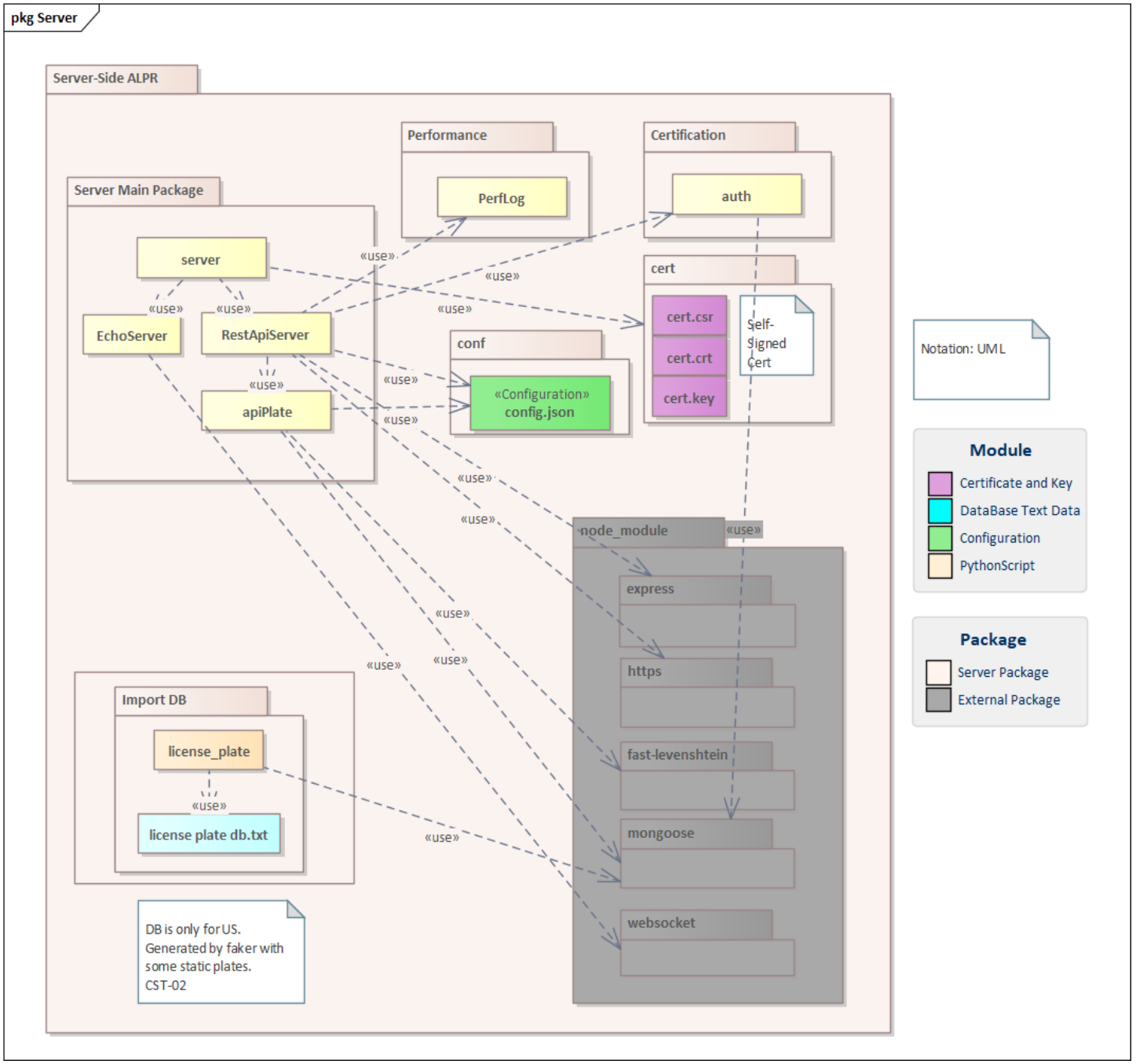
Rationale

1. ADR 3: Use Pipe-and-Filter and manage sampling rate tactic for UI preview
2. ADR 6: Use Facade pattern on C++ openALPR
3. ADR 13: Use Maximum Likelihood Voting Technique to improve the accuracy

Related views

1. Top-level ALPR module uses view

Refinement Server-side module uses view



Element Catalog

Element	Property	Responsibility
Server Main Package		handles plate number queries, and supports authentication/ authorization.
Server		enables RestApiServer / EchoServer
RestApiServer	process REST API	receives REST API from Client and calls operation
EchoServer		provides echo server for the ping from the client
Auth	userFind	manages authorization with id/password from the client

apiPlate	find license plate	finds exact/partial match for license plate number(FR-22)
PerfLog	addPerformanceLog	stores server's performance log (FR-12) - user id - requested plate number - start time - end time - matching result (exact, partial, no match)
	performanceSummary	summarizes performance log
conf	Configuration	server configuration (FR-02) - max_dist_levenshtein - max_num_of_partial_match - jwt_token_expiration
License plate db.txt	Plate information	generated by Faker python script. The generated license plate is only a US plate. (CST-02)
cert	Certification	self-signed certificates (CST-04, CST-06)

Element Interfaces

N/A

Element Behavior

N/A

Variability Guide

N/A

Rationale

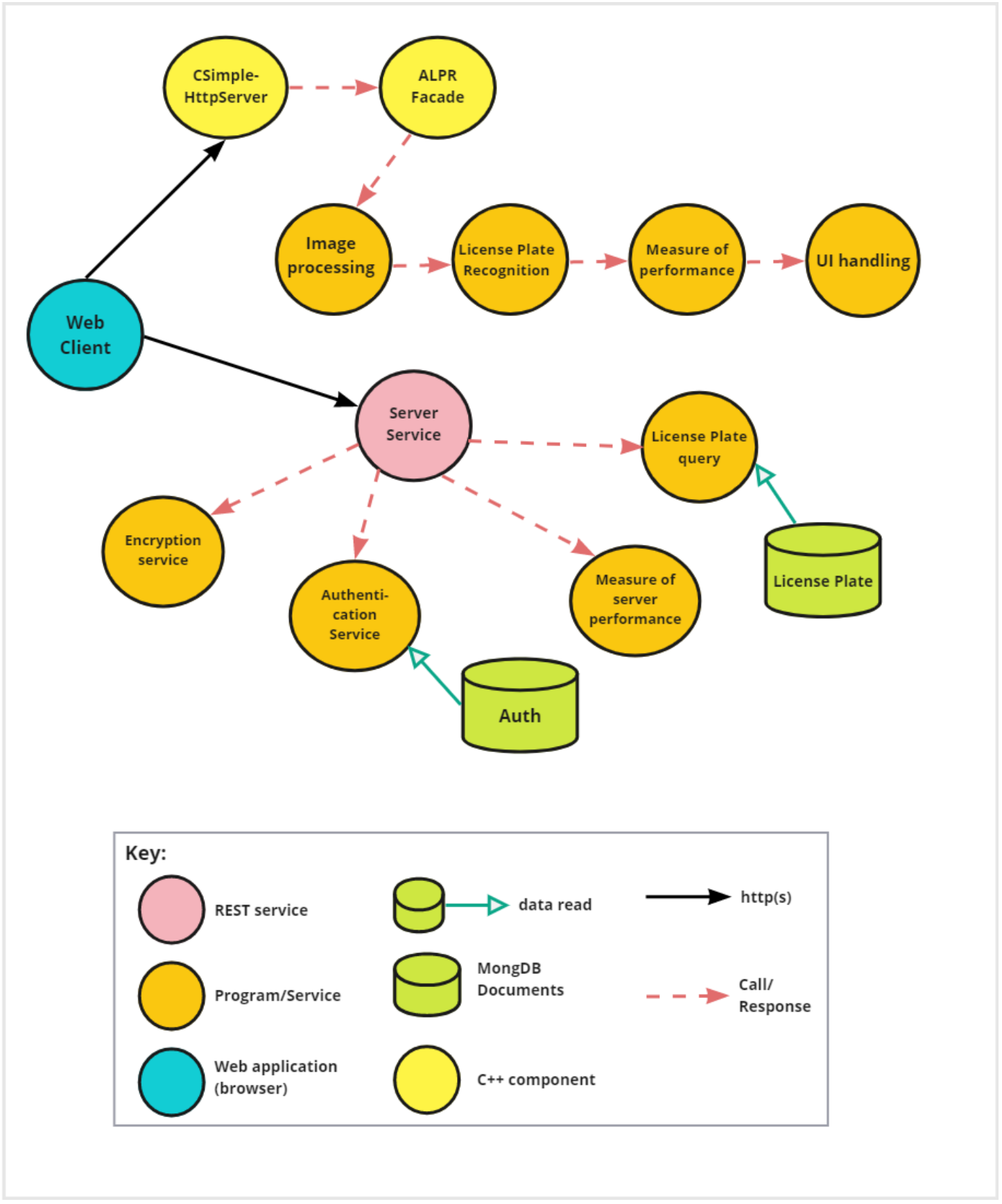
1. ADR 7: Use Node.js Express Server
2. ADR 8: Use Chaining middleware for server's RESTful APIs

Related views

1. Top-level ALPR module uses view

Dynamic perspective - C&C Views

Top-level ALPR Client-Server view



Element Catalog

Element	Property	Responsibility
Web Client	React application	provides a UI and communicates with the server
Encryption Service	TLS(HTTPS)	provides TLS-based secure connection for CST-07

Elements Interfaces

N/A

Elements Behavior

N/A

Variability Guide

N/A

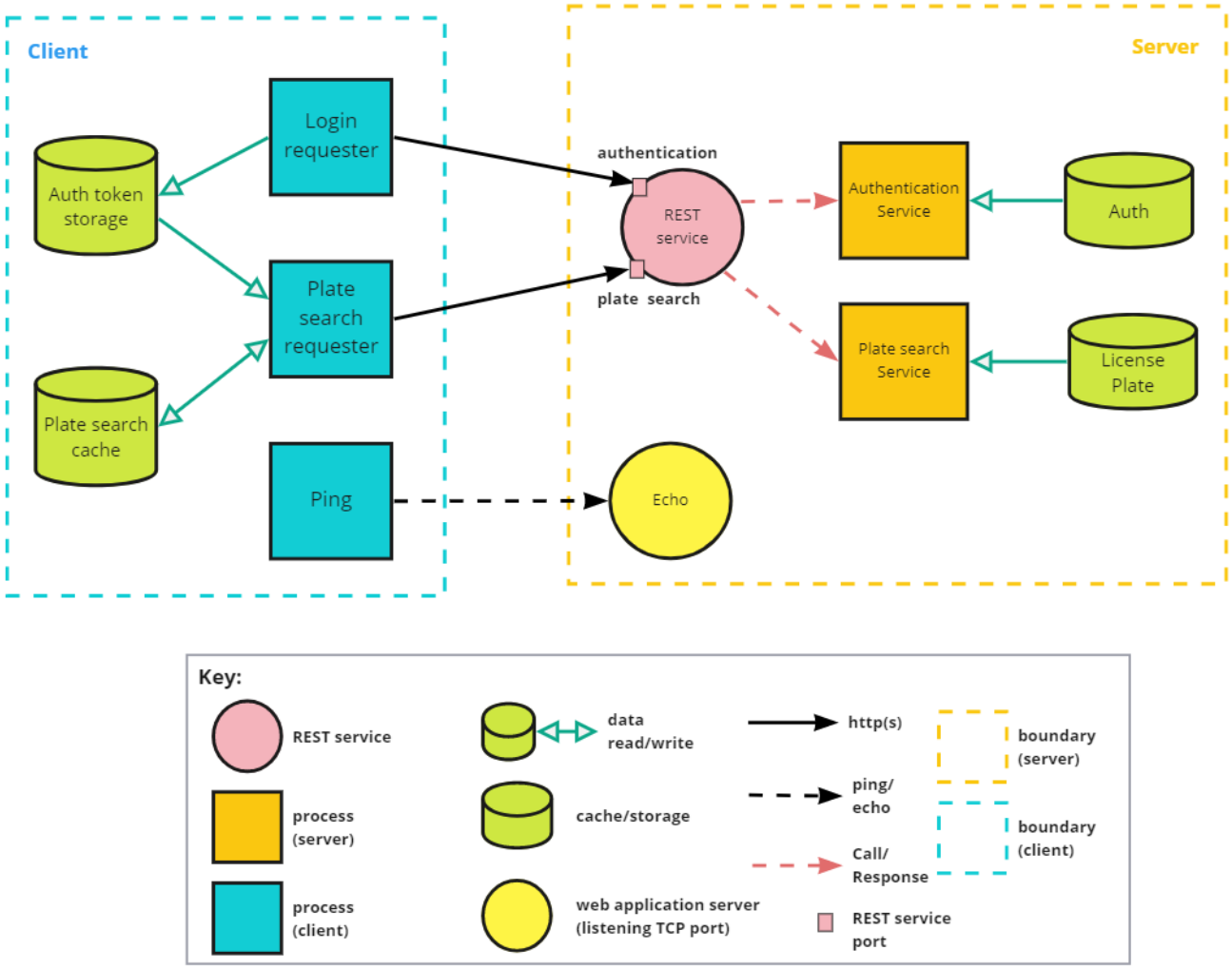
Rationale

1. ADR 9: Use Detect intrusion and authorize users tactics to improve security on RESTful communication between client and server

Related views

1. Refinement Authenticated & encrypted reliable client-server view
2. Refinement ALPR engine to UI preview pipe-and-filter view
3. Refinement Plate number query pipe-and-filter view

Authenticated & encrypted reliable client-server view



Element Catalog

Element	Responsibility
Login requester	provides user authentication and get the token for further authorization
Auth service	provides REST service for user authentication and returns the token for authorization
Auth storage	storage containing user authorization information(ID/ PASSWORD)
Plate search requester	requests to search for a plate number via REST API to server with authentication token and caches the search request and response
Plate search service	provides REST service for searching a plate number
Ping/echo	detects the network communication failure using periodical ping/echo

Elements Interfaces

REST API between React front-end and server

Purpose	URI	HTTP Method	Body	Return
Get car information with plate number	/platenumber/{platenumber}	GET	N/A	[{ "plate":"LKY1360", "status":"Owner Wanted", "registration":"08/22/2023", "ownerName":"Jennifer", "ownerBirth":"11/11/2011", "ownerAddress":"5938", "ownerCity":"West", "vehicleYear":2014, "vehicleMaker":"Chrysler", "vehicleModel":"Tucson", "vehicleColor":"lime", }, …]

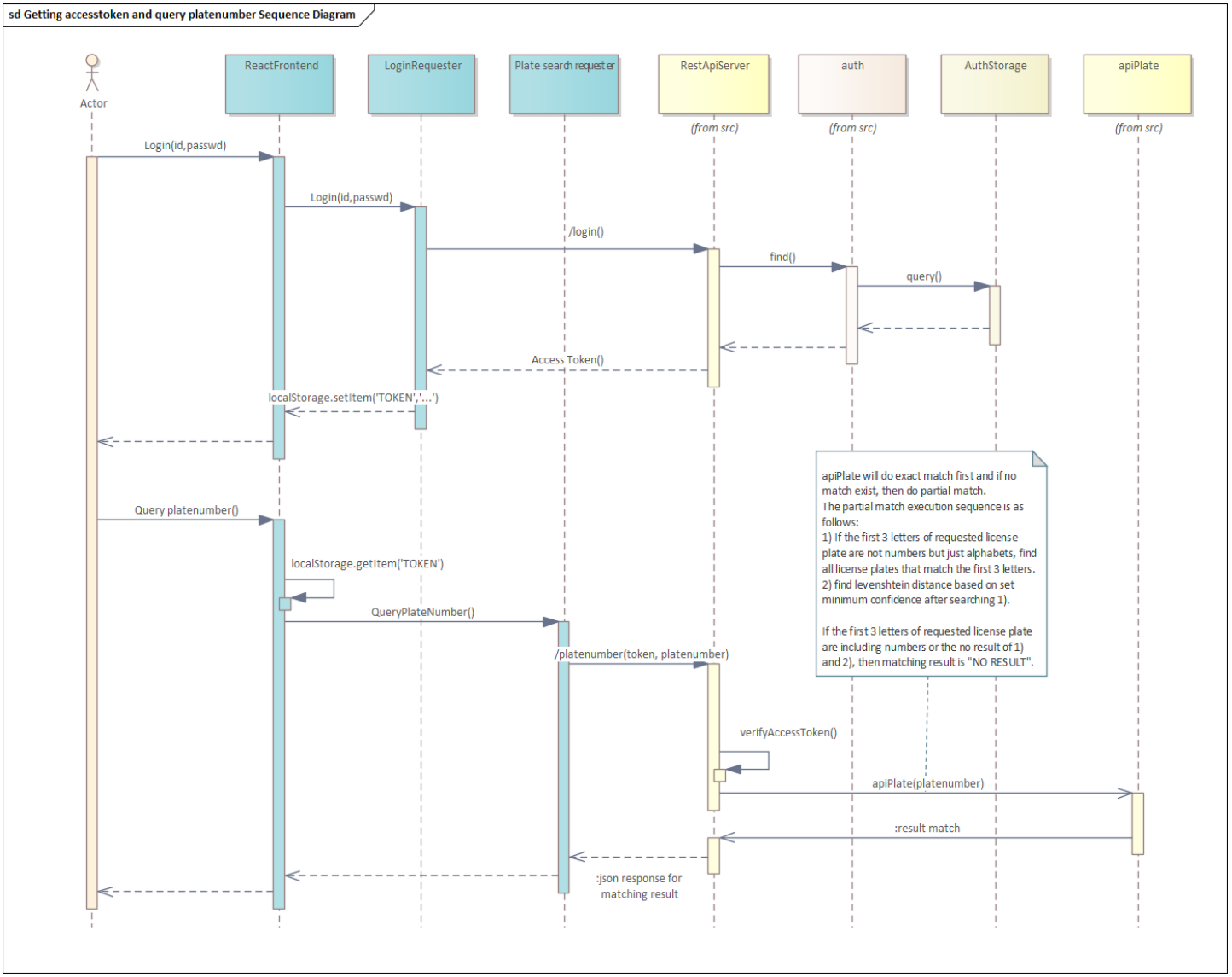
Get authentication	/login	POST	{
			"id": "user",
			"pw": “pass”
			}
				{
				"accessToken": "e11d81ce…",
				"refreshToken":"eyJhbGci…”
				} //Return JWT

notify ALPR start time				
Get server performance metrics	/performance	GET	N/A	[{ "id": "user1", "number_total_query": 1, "number_exact_match": 0, "number_partial_match": 1, "number_no_match": 0 }, …]

Elements Behavior

Sequence diagram to query plate number after authorization.

1. Actors have to login with id and password, and actors get the access token.
2. After getting the access token,actors can use the RESTFul API to query plate numbers.



Variability Guide

N/A

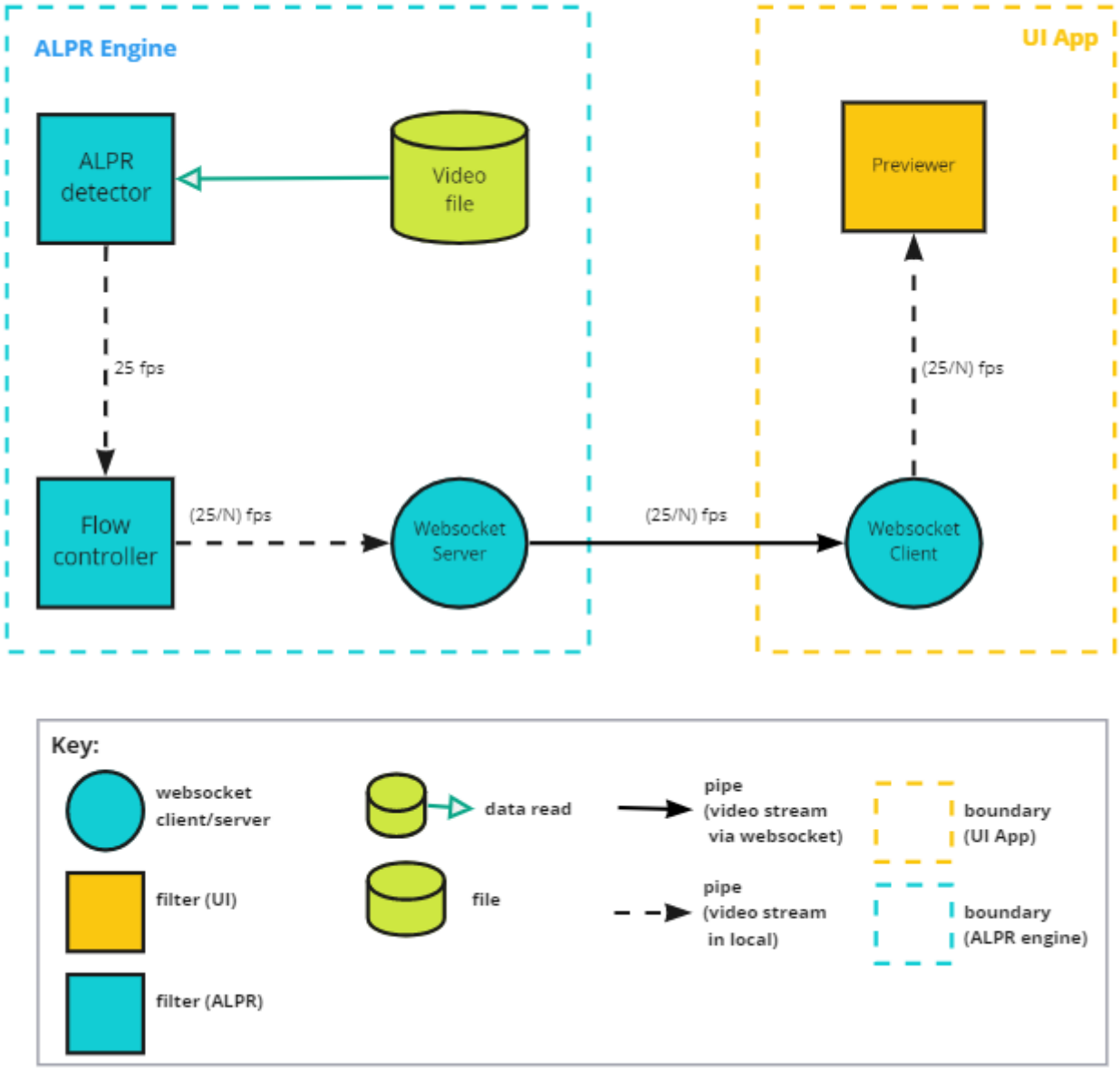
Rationale

1. ADR 8: Use Chaining middleware for server's RESTful APIs
2. ADR 9: Use Detect intrusion and authorize users tactics to improve security on RESTful communication between client and server
3. ADR 4: Use Websocket with ping/echo tactic for network failure
4. ADR 11: Use Levenshtein after pattern matching for Partial Match

Related views

1. Top-level ALPR Client-Server view

ALPR engine to UI preview pipe-and-filter view



Element Catalog

Element	Responsibility
Flow controller	Provides the flow control for video streaming The configuration of the flow control can be updated FR-02

Elements Interfaces

Websocket Interface Protocol between React UI and C++ ALPR engine

Direction	Type	Payload(JSON)

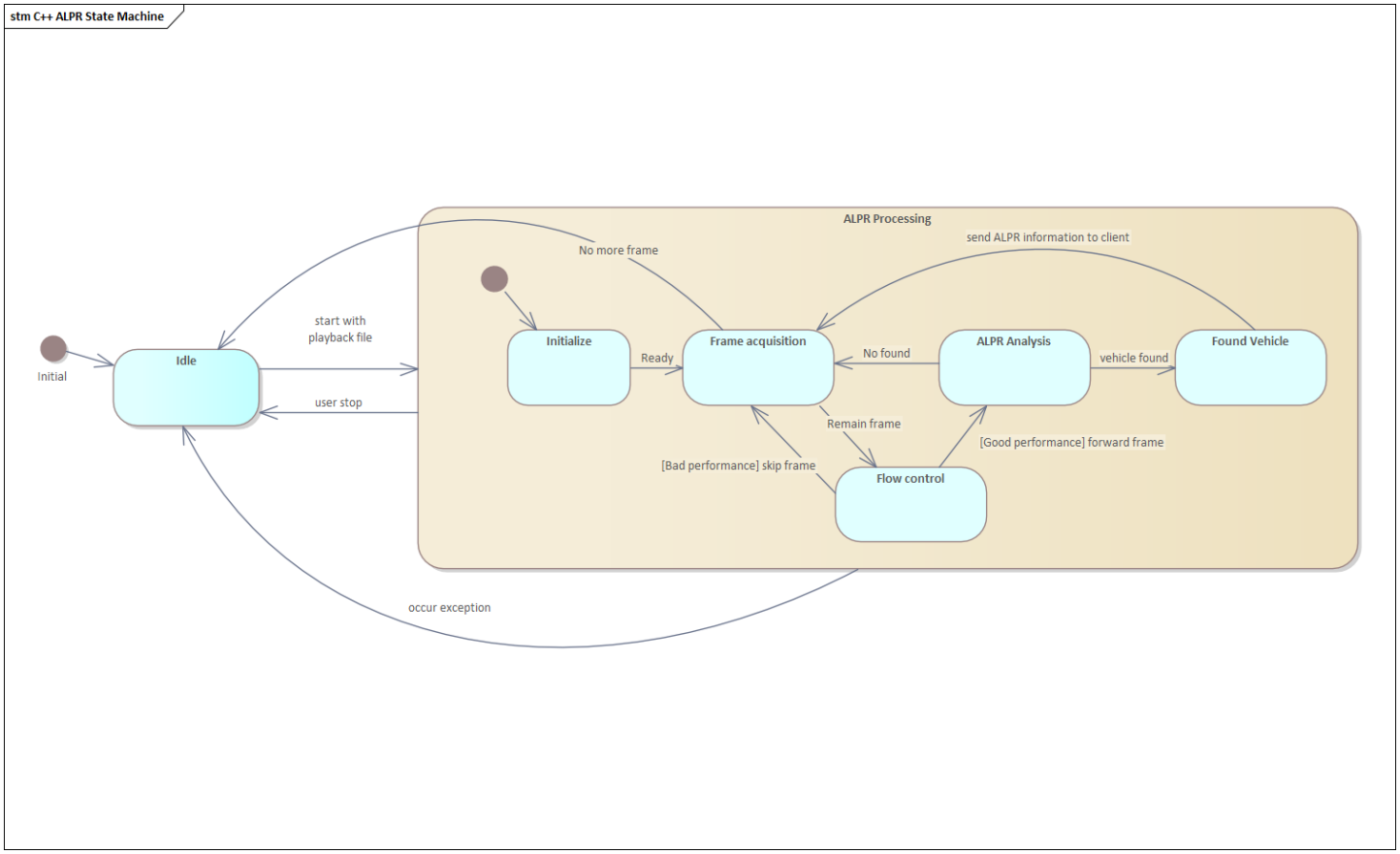
UI → ALPR	connect	by websocket open

	start ALPR	{
		"request" : "start",
		"filepath" : "F:\SWARCH\beaver1a.avi",
		"interval": "25"
		}

	stop ALPR	{ "request" : "stop"}
UI ← ALPR	preview image data	{ "JPEG":"/9j/4AAQSkZJRgABAQAAFB0……" }
	result of ALPR	{ "PLATE": "LKY1360" }
	status	{ "status": "finished" }

Elements Behavior

ALPR engine has state machines to process given video footage's ALPR



Variability Guide

N/A

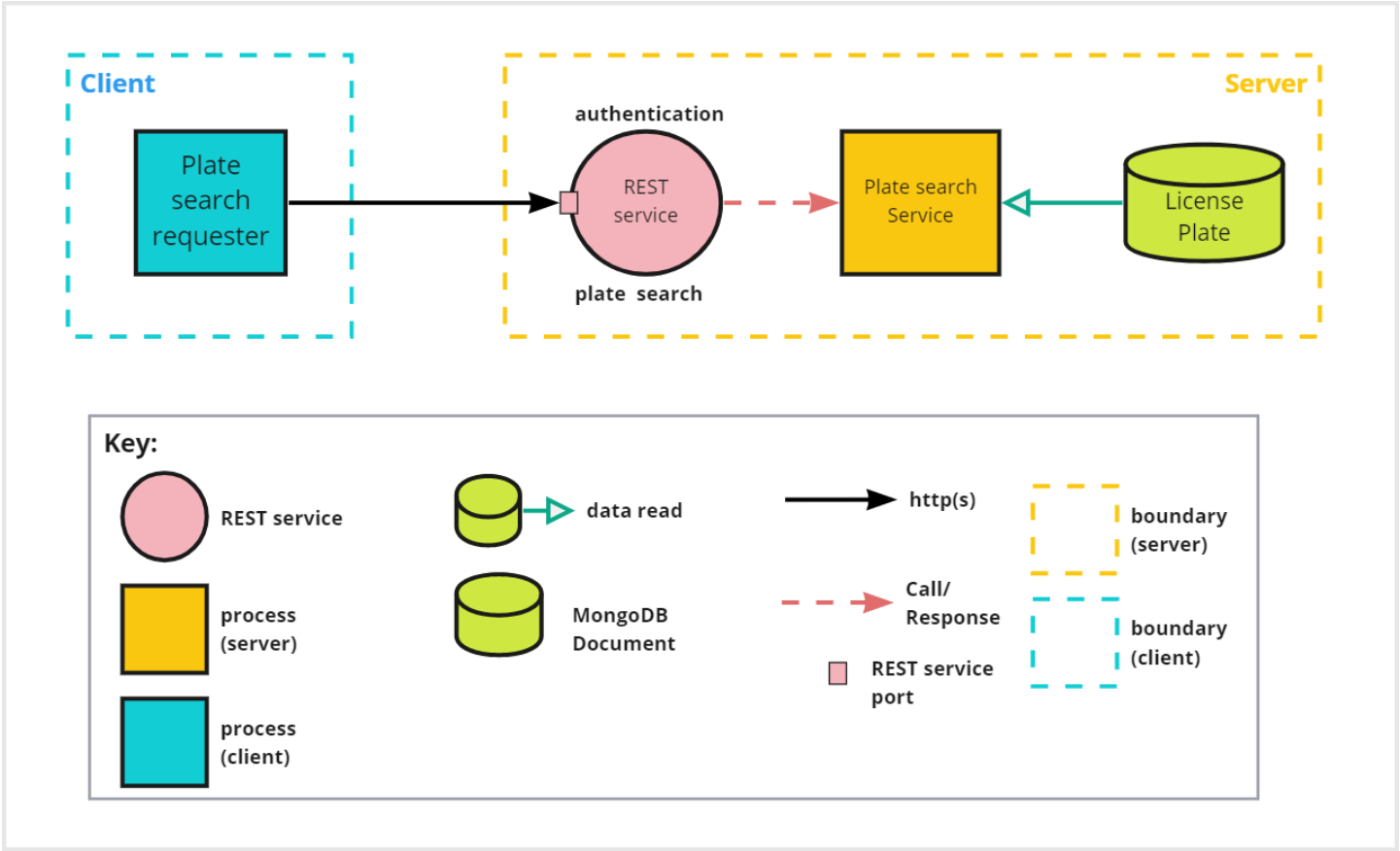
Rationale

1. ADR 3: Use Pipe-and-Filter and manage sampling rate tactic for UI preview

Related views

1. Top-level ALPR Client-Server view

Plate number query pipe-and-filter view



Element Catalog

Element	Responsibility
Plate search requester	requests to search for a plate number via REST API to server with authentication token and caches the search request and response
Plate search service	provides to exact match or partial match with a plate number Exact match - Search via MongoDB Partial Match - Use fuzzy matches for plate numbers and it can get candidate lists of cars. - It calculates levenshtein distance with the candidates list and filters them with thresolder. **If detection time is over 1 second, send “NO MATCH” to client**

Elements Interfaces

REST API between React front-end and server

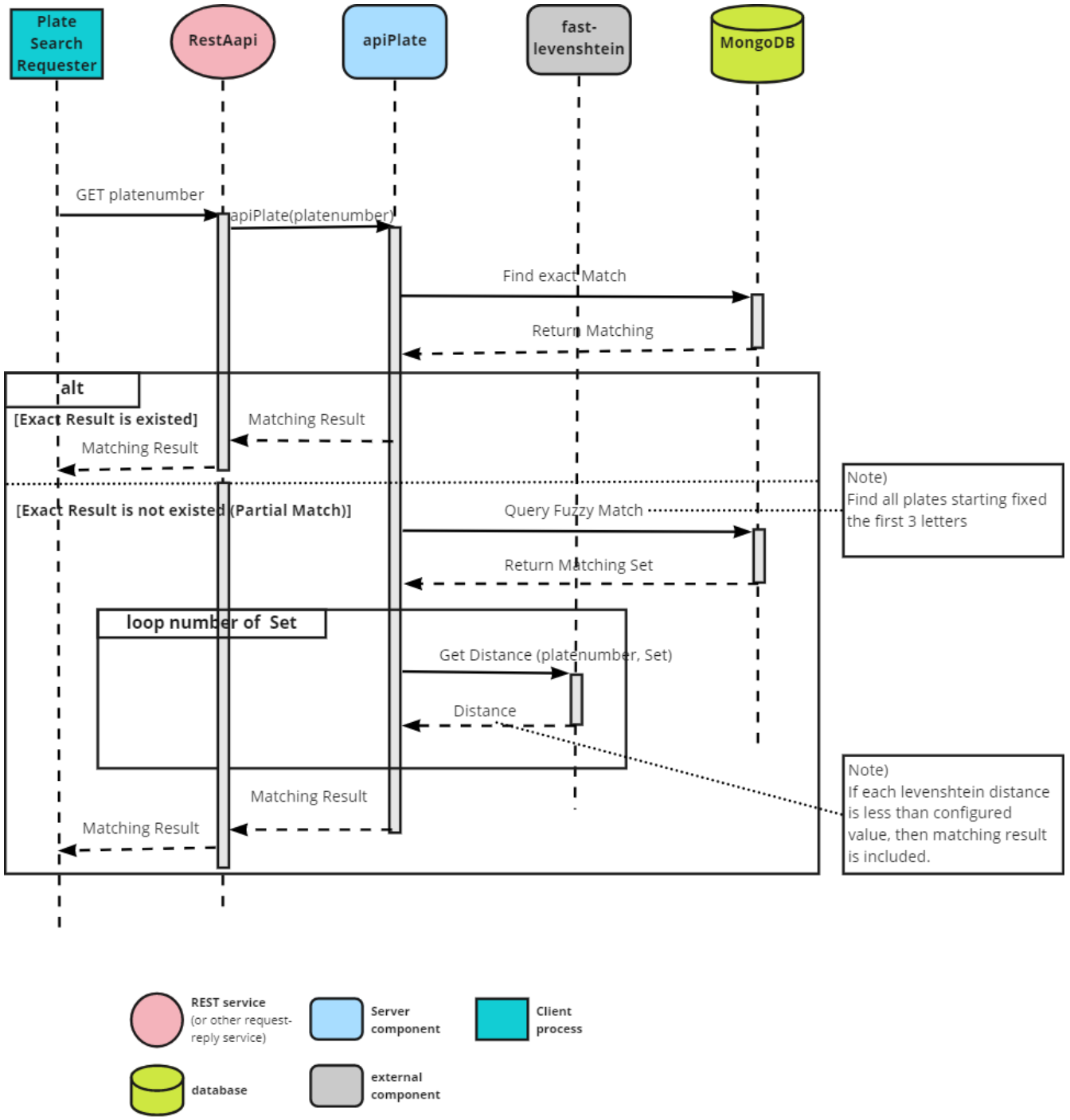
Purpose	URI	HTTP Method	Body	Return

Get car information with plate number	/platenumber/{platenumber}	GET	N/A	[{ "plate":"LKY1360", "status":"Owner Wanted", "registration":"08/22/2023", "ownerName":"Jennifer", "ownerBirth":"11/11/2011", "ownerAddress":"5938", "ownerCity":"West", "vehicleYear":2014, "vehicleMaker":"Chrysler", "vehicleModel":"Tucson", "vehicleColor":"lime", }, …]

Elements Behavior

Sequence diagram to query car information with plate number.

1. It tries to get exact matches with plate numbers.
2. In case of no exact matches, it uses fuzz matches for plat numbers and it can get candidate lists of cars.
3. It calculates levenshtein distance with the candidates list and filters them with thresolder.



Variability Guide

N/A

Rationale

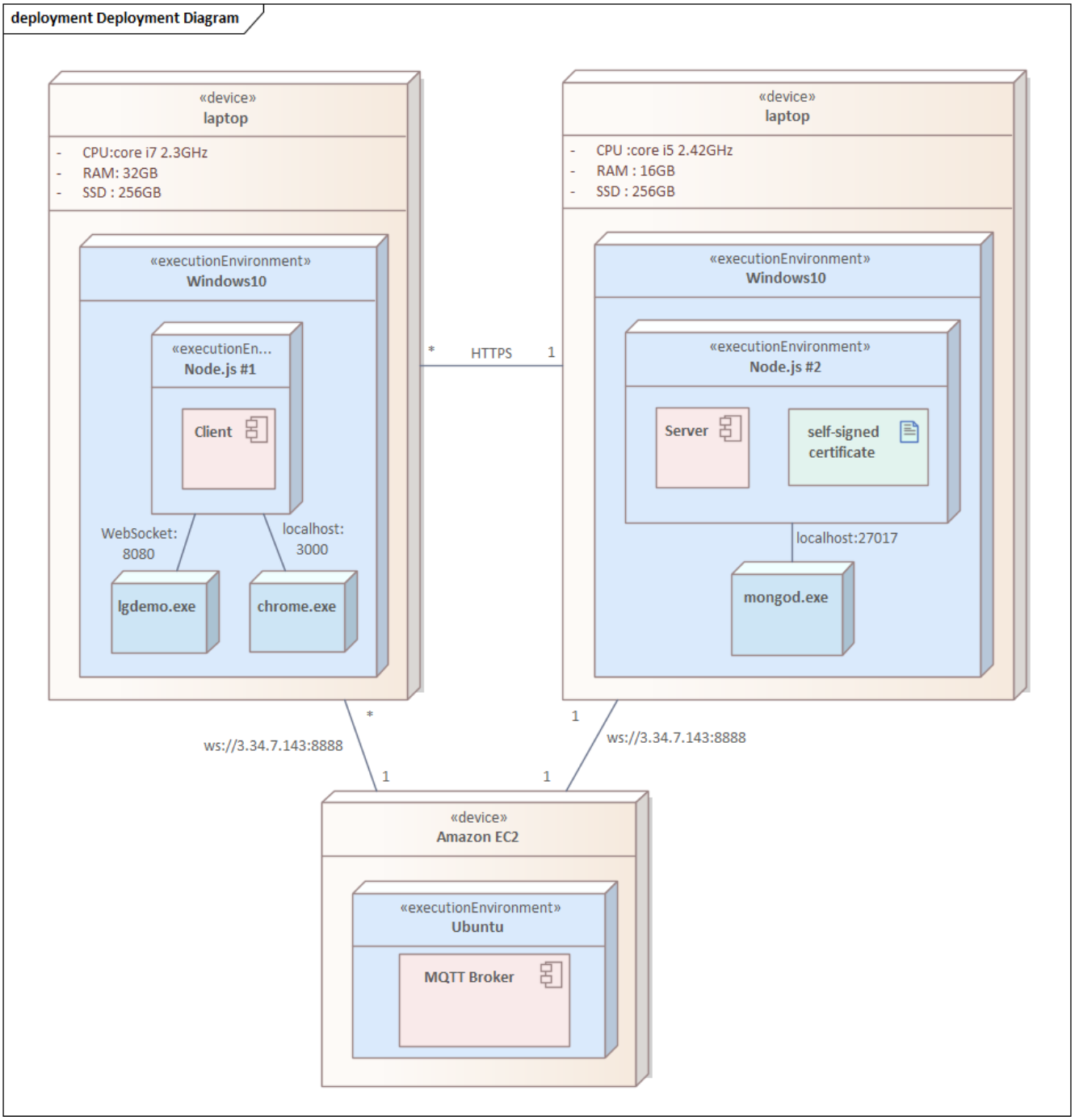
1. ADR 10: Use MongoDB for searching license plate
2. ADR 11: Use Levenshtein Distance after pattern matching for Partial Match

Related views

1. Top-level ALPR Client-Server view

Physical perspective - Deployment Views

Overall ALPR deployment view



Element Catalog

Element	Property	Responsibility
Node.js #1	JavaScript runtime	Node.js® is a JavaScript runtime. Node.js supports running React app.
Client	RESTful API	Client is responsible for http communication with the server.

	client	CST-01
lgdemo.exe	license plate recognition engine	lgdemo.exe is responsible for recognizing the license plate in the frame. It sends a frame marked with a license plate to the Client via WebSocket.
chrome.exe	rendering engine	chrome.exe is responsible for rendering playback and showing plate number related information.
Node.js #2	JavaScript runtime	Node.js® is a JavaScript runtime. Node.js is designed to build scalable network applications.
Server	web application service	Server is a web application service that provides RESTful APIs related to authentication, authorization, license plate lookup and query related information. CST-01
self-signed certificate	certificate	self-signed certificates encrypt communication between the client and server.
mongod.exe	Database	MongoDB has 25 million license plate data. it provides a query.
MQTT broker	Message broker	MQTT broker used for publish-subscribe pattern. MQTT broker is to filter messages based on topic, and then distribute them to subscribers.

Elements Interfaces

N/A

Elements Behavior

N/A

Variability Guide

You can configure following items by editing and restart the client application (client.conf)

Configurable Item	Description	Usage
frame_skip	Adjust frame skip rates on laptop to render application preview	frame_skip = [1..20]

You can configure following items by editing and restart the server application (conf.json)

Configurable Item	Description	Usage
max_dist_levenshtein	Set maximum levenshtein distance to support a partial match	max_dist_levenshtein = {1| 2| 3| …}
expiration	Set expiration time for log-in session	expiration = {1|2|3|4}

Rationale

1. ADR 2: Use React Front-end
2. ADR 5: Use C++ openALPR as the license plate recognition engine
3. ADR 7: Use Node.js Express Server
4. ADR 10: Use MongoDB for searching license plate
5. ADR 9: Use Detect intrusion and authorize users tactics to improve security on RESTful communication between client and server

Related views

N/A

Technical Experiments

Experiment 1: DB performance test

Results and recommendations

We decided to use MongoDB because MongoDB can handle 25M records and execute exact matching in the fastest time among the DBs tested. MongoDB only supports partial match using levenshtein distance on the cloud version named Atlas, but we decided to use pattern matching as mentioned in Experiment 6.

Objective

This experiment is to select DB or DB technology we will use.

we have requirements

1. (QA-04) The system should respond to plate number detection requests within 1 second when DB has 25,000,000 records for car detection.

We will find out which DB is best fit for the studio project.

Status

[Planned | In progress | Suspended | Canceled | **Concluded**]

Expected outcomes

After this experiment we get the following test report.

Lucene is a little bit slower than MongoDB, but it supports partial match.

		MariaDB	**MongoDB**	**Solr/Lucene**
Performance	**Can handle 25,000,000 rows**	O	O	O
	Query latency on 25,000,000 rows (exact match)	< 40ms	< 10ms	< 3~100ms. (Include Partial Match)

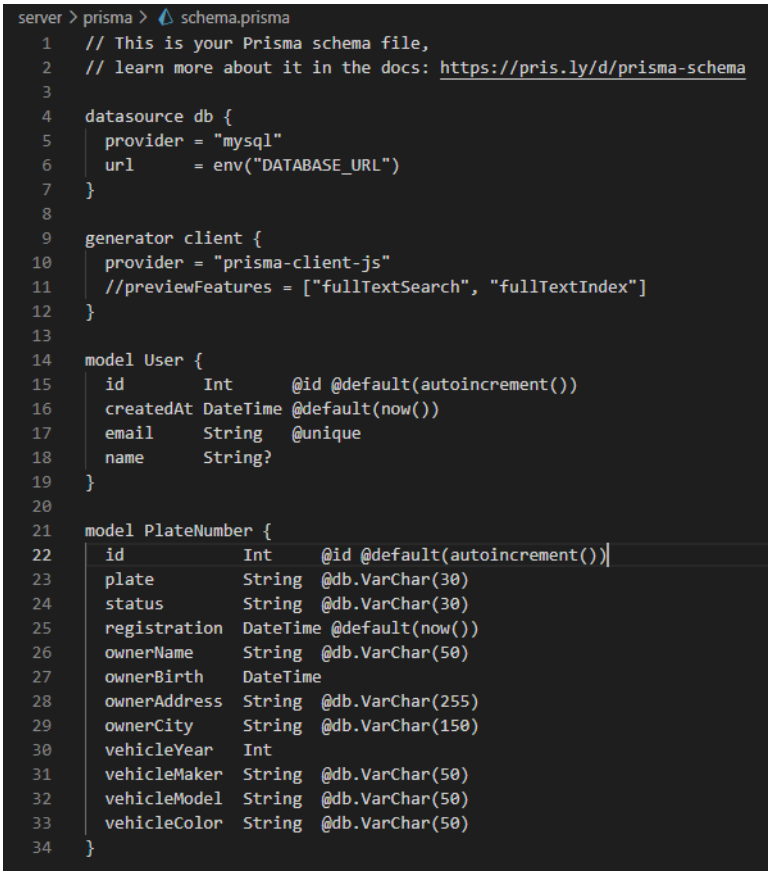
Modifiability	**Integrate with server framework**	Easily Modified by providing npm package	Easily Modified by providing npm package	Easily Modified by using Solr Command
Support Partial match		O (Support partial match as creating SQL levenshtein function)	O (Support only on cloud version named Atlas)	O (Lucene support fuzzy search based on edit (Levenshtein) distance)

Resources required

1. Software tools
    - constructed and testing DB through Node.js on Windows 10
    - MariaDB
        - Install MariaDB 10.6
        - HeidiSQL
    - MongoDB
        - MongoDB 5.0.9
        - MongoDB Compass (MongoDB Connection / creating index and using mongo shell)
    - Solr
        - TBD
    - Insomnia - REST API Tool for DB Performance Test
2. Frameworks
    - Maria DB
        - prisma package on Node.js
    - Mongo DB
        - mongoose package on Node.js
    - Solr
        - TBD
3. Record generator like Faker and DB importer
    - Generated 25M plate records using the provided Faker generator python script and imported into MariaDB, MongoDB and Solr DB, respectively.
4. 2 person’s 15 days effort

Experiment description

- Research on the Internet for alternatives
    - We planned to experiment with MariaDB with SQL, MongoDB with NoSQL and Solr/Lucene in our lecture.
- Build record generator and DB Importer
    - MariaDB import
        - Create a Prisma schema as shown below for Prisma Migrate.

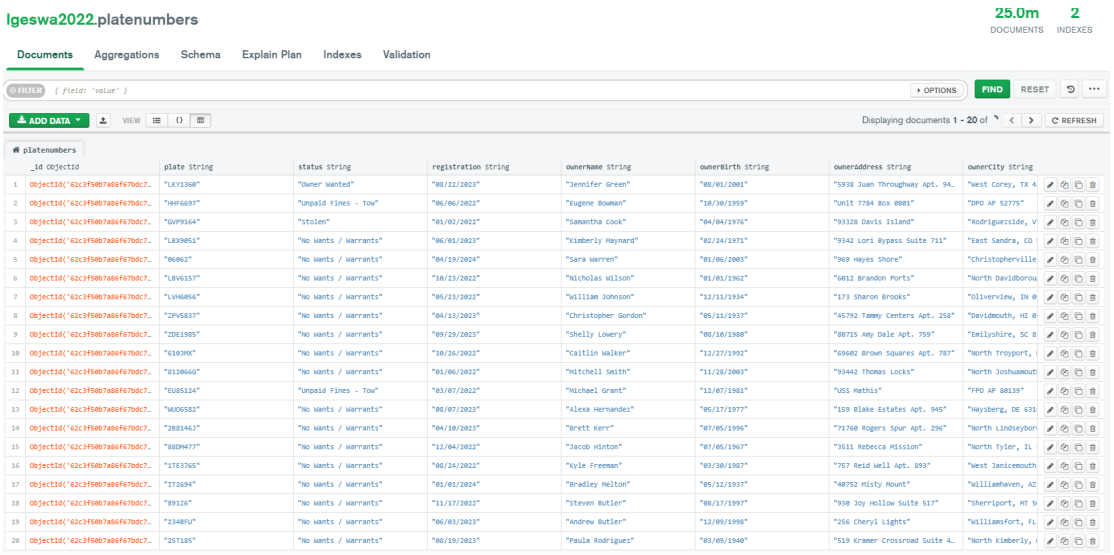


```
server > prisma > schema.prisma
1   // This is your Prisma schema file,
2   // learn more about it in the docs: https://pris.ly/d/prisma-schema
3
4   datasource db {
5     provider = "mysql"
6     url      = env("DATABASE_URL")
7   }
8
9   generator client {
10    provider = "prisma-client-js"
11    //previewFeatures = ["fullTextSearch", "fullTextIndex"]
12  }
13
14  model User {
15    id        Int      @id @default(autoincrement())
16    createdAt DateTime @default(now())
17    email     String   @unique
18    name      String?
19  }
20
21  model PlateNumber {
22    id            Int      @id @default(autoincrement())
23    plate         String   @db.VarChar(30)
24    status        String   @db.VarChar(30)
25    registration  DateTime @default(now())
26    ownerName     String   @db.VarChar(50)
27    ownerBirth    DateTime
28    ownerAddress  String   @db.VarChar(255)
29    ownerCity     String   @db.VarChar(150)
30    vehicleYear   Int
31    vehicleMaker  String   @db.VarChar(50)
32    vehicleModel  String   @db.VarChar(50)
33    vehicleColor  String   @db.VarChar(50)
34  }
```

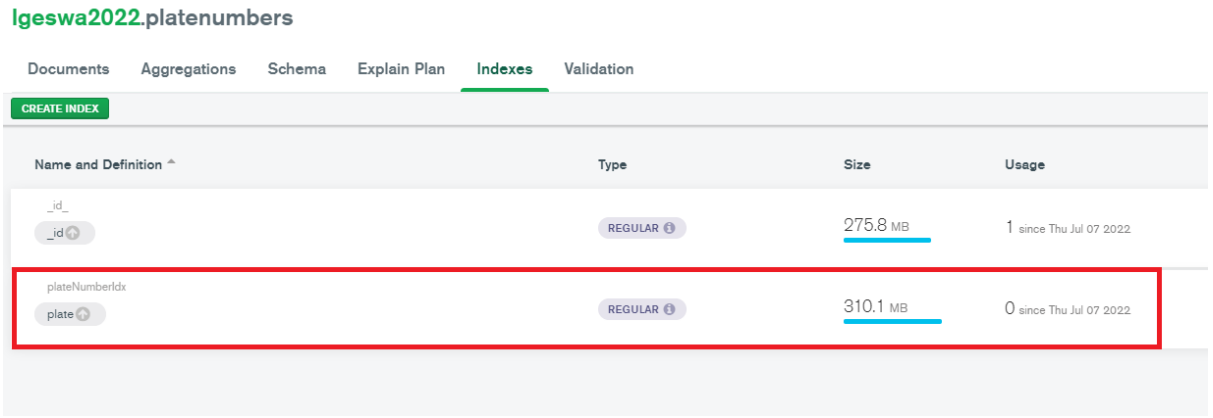
        - After migration, the 25M plate records file is parsed and imported into MariaDB.
        - created index through SQL query for plate in imported MariaDB as below:

          CREATE INDEX idx_platenumber ON platenumber (plate);

    - MongoDB import
        - created and ran a python script for importing into MongoDB.



- created index for plate in imported MariaDB as below:

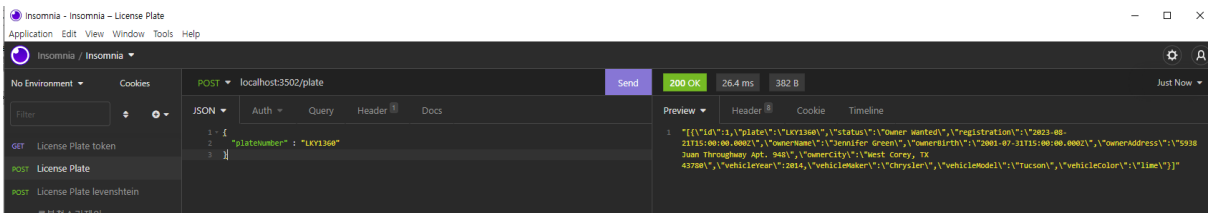


- Running tests
  - MariaDB Test
    - Created apiPlate.js for MariaDB

server > src > JS apiPlate.maria.js > apiPlate

```
1   import { PrismaClient } from '@prisma/client';
2   import levenshtein from 'fast-levenshtein';
3
4   var result;
5
6   const prisma = new PrismaClient();
7
8   const apiPlate = async (req, res) => {
9     try {
10      console.log('[apiPlate] req.body:', req.body);
11      const plateNumber = req.body.plateNumber;
12      console.log('[apiPlate] plateNumber: ' + plateNumber);
13
14
15      // exact match
16      var result = await prisma.plateNumber.findMany({
17        where: {
18          plate: plateNumber,
19        },
20      });
21
22      if (result.length == 0) //if no match
23      {
24        //TODO: partial match
25      }
26
27      console.log(result);
28      res.json(JSON.stringify(result));
29    } catch (err) {
30      console.error(err);
31      res.status(500);        //sending failure to the client
32    } finally {
33      await prisma.$disconnect();
34    }
35  }
```

- Run REST API using insomnia



- Check duration time on terminal after REST API

```
13
14      let start = new Date();
15      // exact match
16      var result = await prisma.plateNumber.findMany({
17        where: {
18          plate: plateNumber,
19        },
20      });
21      let end = new Date();
22      console.log(`${end - start}ms`)
23
24      if (result.length == 0) //if no match
25      {
26        //TODO: partial match
27      }
28
29      //console.log(result);
30      res.json(JSON.stringify(result));
31    } catch (err) {
32      console.error(err);
33      res.status(500);        //sending failure to the client
34    } finally {
35      await prisma.$disconnect();
```

문제 출력 디버그 콘솔 터미널

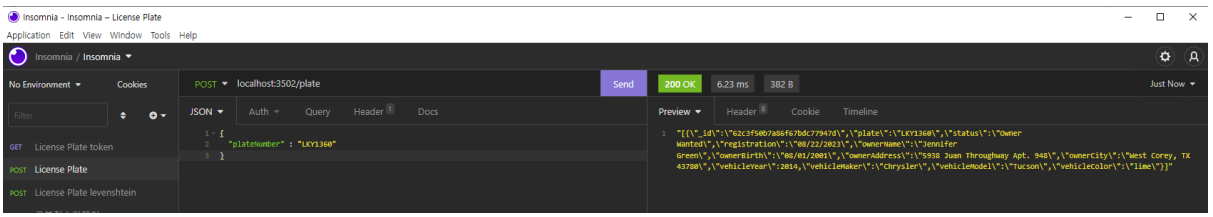
```
26ms
POST /plate 200 28.743 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
21ms
POST /plate 200 22.391 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
32ms
POST /plate 200 33.356 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
23ms
POST /plate 200 24.758 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
23ms
POST /plate 200 24.706 ms - 382
```

- MongoDB Test
  - Created apiPlate.js for MongoDB

server > src > JS apiPlate.js > platenumberSchema > registration

```
62  // 5. 연결 성공
63  db.once('open', function() {
64    console.log('Connected!');
65  });
66  const collections = mongoose.model('platenumber', platenumberSchema);
67
68  const apiPlate = async (req, res) => {
69    try {
70      let distance;
71      var partial_result = [];
72      console.log('[apiPlate] req.body:', req.body);
73      const plateNumber = req.body.plateNumber;
74      console.log('[apiPlate] plateNumber: ' + plateNumber);
75
76      let start = new Date();
77      // exact match
78      var result = await collections.find(
79        {
80          plate: plateNumber
81        }
82      );
83      let end = new Date();
84      console.log(end - start)
85
86      if (result.length)
87      {
88        res.json(JSON.stringify(result));
89      }
90      else //if no match
91      {
92        let word = plateNumber.substring(0, 3);
93        console.log(word);
94        //TODO: partial match
95        var result = await collections.find(
```

- Run REST API using insomnia



- Check duration time on terminal after REST API

```
70      let distance;
71      var partial_result = [];
72      console.log('[apiPlate] req.body:', req.body);
73      const plateNumber = req.body.plateNumber;
74      console.log('[apiPlate] plateNumber: ' + plateNumber);
75
76      let start = new Date();
77      // exact match
78      var result = await collections.find(
79        {
80          plate: plateNumber
81        }
82      );
83      let end = new Date();
84      console.log(`${end - start}ms`)
85
86      if (result.length)
87      {
88        res.json(JSON.stringify(result));
89      }
90      else //if no match
```

문제 출력 디버그 콘솔 터미널

```
3ms
POST /plate 200 7.034 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
3ms
POST /plate 200 4.402 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
3ms
POST /plate 200 5.123 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
3ms
POST /plate 200 3.715 ms - 382
[apiPlate] req.body: { plateNumber: 'LKY1360' }
[apiPlate] plateNumber: LKY1360
3ms
POST /plate 200 4.680 ms - 382
```

- Solr Test
  - TBD

Duration

Experiment duration : 27 Jul ~ 15 Jun

Links and references

DB record sample is provided with Initial Project files..

prisma package : https://www.npmjs.com/package/prisma
mongoose package : https://www.npmjs.com/package/mongoose

prisma schema : https://www.prisma.io/docs/concepts/components/prisma-schema

Experiment 2: Response Time Performance Test

Results and recommendations

We planned the experiment to select a Node.js framework for best performance in response time. After searching google and investigating Node.js framework, we found that “express” is the most popular and universal. Additionally, express provides lightweight, powerful tools for HTTP servers. So this experiment was canceled and we decided to select and use “express” for Node.js framework.

Objective

This experiment is to select a Node.js framework for the server. We will look for a Node.js framework for best performance in response time.

we have requirements

1. (QA-04) The requests for car information with plate number should be executed within 1 second.

We will find out which Node.js framework is best fit for the studio project.

Status

[Planned | In progress | Suspended | **Canceled** | Concluded]

Expected outcomes

After this experiment we get the following test report.

	express	sails	meteor
Performance			
Productivity			
Security			

Resources required

N/A

Experiment description

N/A

Duration

N/A

Links and references

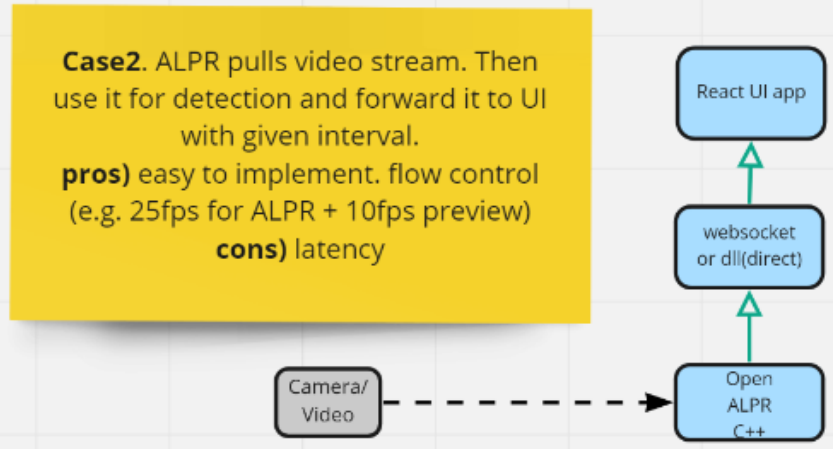
The Best NodeJS Frameworks for 2021 :
https://rapidapi.com/blog/best-nodejs-frameworks/

7 MOST POPULAR NODE.JS FRAMEWORKS IN 2019:
https://x-team.com/blog/most-popular-node-frameworks/

Experiment 3: Streaming performance test for UI and ALPR

Results and recommendations

We decided to let the video streaming first goes into the ALPR engine and then the ALPR engine sends the result as the preview for the UI app as follow:



To get the decision, we reviewed four potential cases and measured the performance for two cases among them. Finally, we got a decision based on the following performance result.

Measurement		UI to ALPR	ALPR to UI
detection	fps	less than 25 fps (or too much delay)	25 fps
	image quality	480p	480p
preview	fps	25 fps	25 fps (or less; flow control)
	image quality	less than 480p (or too much delay)	480p

Objective

- The UI and the ALPR engine are separated in our system, and we have a requirement that the video stream from a webcam or a file must be shown as a preview on the UI while the ALPR engine is detecting the license plate recognition.
- The test is to determine how to handle video streams for the UI preview and the ALPR engine simultaneously.

- This test affects the design for the UI and the ALPR engine.

Status

[Planned | In progress | Suspended | Canceled | **Concluded**]

Expected outcomes

The design decision and the performance table as follows:

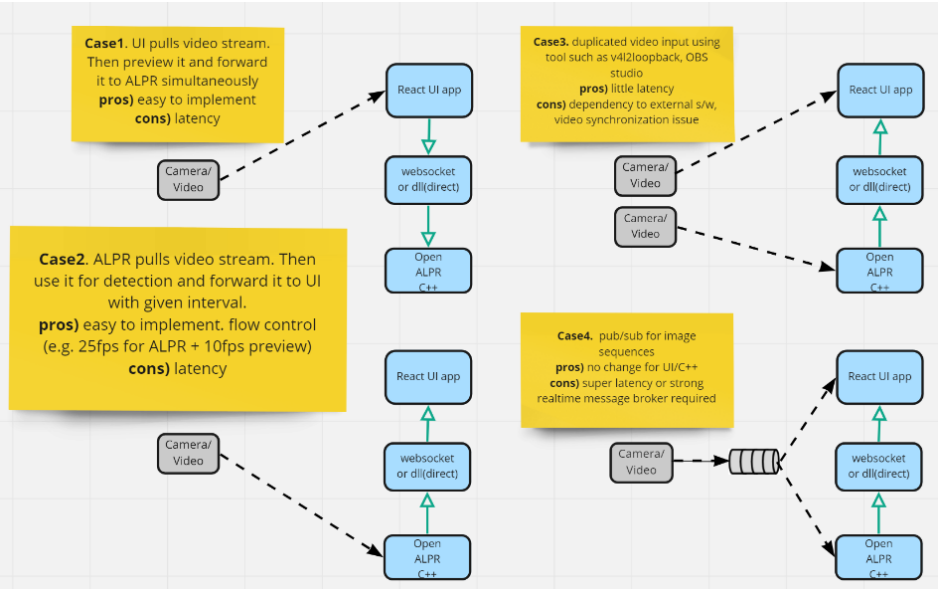
Measurement		UI to ALPR	ALPR to UI.
detection	fps		
	image quality		
preview	fps		
	image quality		

Resources required

1. React UI application and C++ ALPR engine on the same machine.
2. 2 person-days in effort

Experiment description

We considered the following four potential cases in which the video stream is provided to both the UI and the ALPR engine as follows:



- Case3 has been excluded since it increases complexity and brings dependency to external modules(e.g. v4l2loopback, OBS studio) causing unexpected problems.
- Case4 has been excluded since a well-known message broker such as MQTT is too slow to be used for video streaming and other faster message brokers such as ROS2 are too heavy(overhead) to use.
- Therefore, we compared Case1 and Case2 as we implemented prototyping:
    - For Case1, there was too much delay on 25 fps streaming. We had to decrease fps or the image quality which lowers ALPR accuracy.
    - For Case2, the speed(performance) depends on CPU and furthermore, we can do flow-control which sends video streams less than 25 fps for laptops with low CPU capacity.

Duration

Experiment duration : 27 Jun ~ 1 Jul

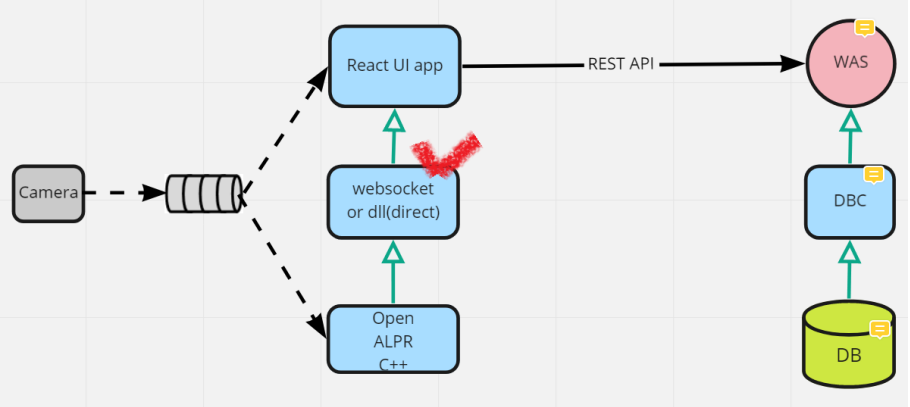
Links and references

N/A

Experiment 4: Communication method test for UI and ALPR

Results and recommendations

We decided to use websocket for the video streaming from the ALPR engine to the UI preview.



To get the decision, we reviewed four potential cases and measured the performance for two cases among them. Finally, we got a decision based on the following performance result.

Case	Pros	Cons
direct call via *dll*	Fast communication	Complicated or not verified
websocket	Easy implementation	Network traffic (can be overcome by flow control)
python wrapping	N/A	Not working or duplicated

Objective

- The test is to determine how to **send** the video streams (processed by the ALPR engine) to the UI preview.
- This test affects the communication method for the UI and the ALPR engine.

Status

[Planned | In progress | Suspended | Canceled | **Concluded**]

Expected outcomes

The decision of which communication method to use and the comparison table:

Case	Pros	Cons
direct call via *dll*		
websocket		
python wrapping		

Resources required

- React UI application and C++ ALPR engine on the same machine.
- A given video file.
- 3 person-days in effort

Experiment description

We derived four potential cases in which video streams provided to UI and ALPR:

1. Research on <direct call via *dl*>
    a. We found some methods for Node.js to use *.dll* on the internet but it seems to be complicated which is bug-prone.
    b. It is also risky because it is not widely accepted(not verified).
2. Research on <websocket>
    a. We made a prototype and found it works properly.
    b. Network traffic could be a problem however it is solved by flow control described in Experiment 3.
3. Research on <python wrapping>
    a. A Python wrapper for OpenALPR is included in the OpenALPR project however it does not work so we abandoned it.
    b. Even if the wrapper is available, we still need to consider the communication method between Python and Node.js.

Duration

Experiment duration : 27 Jun ~ 1 Jul

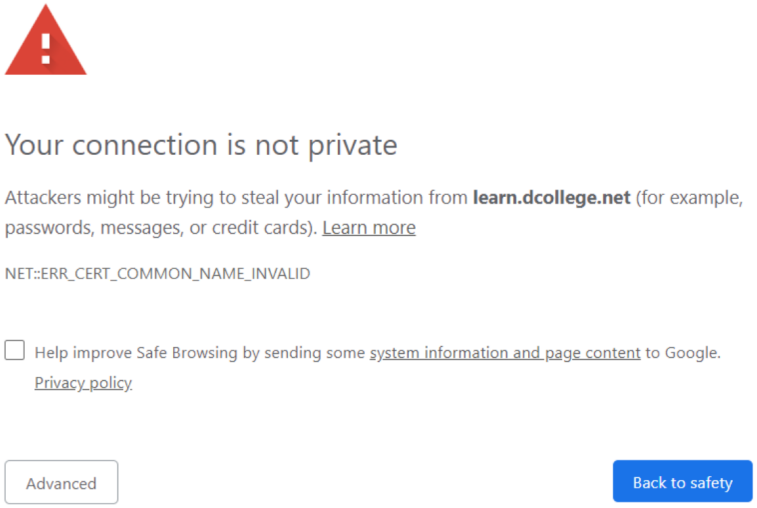
Links and references

N/A

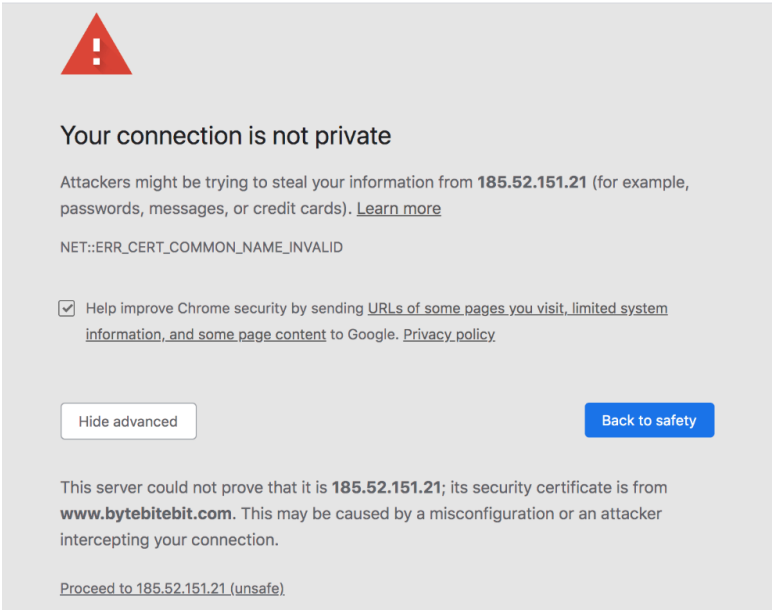
Experiment 5: Install X.509 Certificate

Results and recommendations

Since *WeWork*, where the office we work, does not provide a public IP address for users, we cannot make DNS available for our server. This means that we cannot use officially signed certificate such as *Letsencrypt*.
Instead of using an officially signed certificate, we will generate and use the certificate from *OpenSSL* and install it to the server. However this causes a user to agree to the security warning on the browser by clicking the button at the very beginning, connecting https://localhost:3503/



▼ click <Advanced> button



▼click <Proceed to …> then restart the application.

Objective

X.509 certificates are required to provide TLS(HTTPS) for secure connection [CST-06]. However, a public IP is not provided in *WeWork*, where our server is running, so the certificate cannot be bound. We need to find a way to provide TLS.

Hint: The Certificate Authority failed to download the challenge files from the temporary standalone web server started by Certbot on port 80. Ensure that the listed domains point to this machine and that it can accept inbound connections from the internet.

Status

[Planned | In progress | Suspended | Canceled | **Concluded**]

Expected outcomes

Find a way to set up the X.509 certificate to provide TLS.

Resources required

1. Test environment
2. 2 person’s 3 days effort

Experiment description

Research on the internet.

Duration - N/A

Links and references - N/A

Experiment 6: Partial match test

Results and recommendations

Before proceeding with this experiment, the DB performance test (Experiment 1) was concluded first. So the DB we used is MongoDB. Exceptionally, MariaDB is used in Case 2. We decided to use Case3 (calculating Levenshtein distance after pattern matching) as partial matching. Case3 is the fastest time among the cases we tested.

Objective

This experiment is to make a design decision about how to perform the partial match upon prototyping potential solutions.

we have requirements

1. (FR-22) The system should return the best match license plate if there is not an exact match that includes a configurable minimum confidence threshold to support a partial match.

We will find out which solution is the best partial match for the studio project.

Status

[Planned | In progress | Suspended | Canceled | **Concluded**]

Expected outcomes

After this experiment we get the following test report.

Case	Description	pros	cons
Case 1	Just calculating Levenshtein(a, b) via npm library(https://www.npmjs.com/package/fast-levenshtein)	Operating partial match is possible based on levenshtein distance for all plate numbers in DB	Processing time is too long (Approximately 2.5s per 1000 records)
Case 2	Using prisma (Only MariaDB)	Operating partial match is possible	Processing time is too long

	(https://github.com/prisma/prisma/issues/10439)	based on levenshtein distance for all plate numbers in DB	(Approximately 0.8s per 3000 records)
Case 3	calculating Levenshtein(a, b) via npm library after pattern matching	processing time is greatly reduced and it solves the processing time in Case1. **(Average Processing Time : 150ms)**	The use of pattern matching has a constraint. (Constraint: The first 3 letters must be fixed)

Resources required

1. Software tools
    - Node.js on Windows 10
    - MongoDB
        - MongoDB 5.0.9
        - MongoDB Compass (MongoDB Connection / creating index and using mongo shell)
    - MariaDB
        - MariaDB 10.6
        - HeidiSQL
    - Insomnia - REST API Tool for Partial Matching Test
2. Frameworks
    - fast-levenshtein package
    - Maria DB
        - prisma package on Node.js
    - Mongo DB
        - mongoose package on Node.js
3. 2 person's 5 days effort

Experiment description

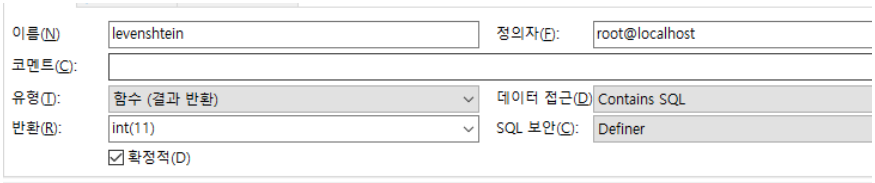
Experiment procedure

- Case 1) Just calculating Levenshtein(a, b) via npm library each index in DB

```
// CASE1 : 1000개당 약 2.5초 소요 (사용불가)
let start, end;
let distance;
start = new Date();
for (var i =1 ;i <= 25000000 ; i++) {
  result = await prisma.plateNumber.findUnique({
    where:
    {
      id: i,
    },
  })

  distance = levenshtein.get(plateNumber, result.plate);
  if (distance < 2)
    console.log(distance);
  //console.log(result.plate);
  //console.log(i);
  if (i%1000==0)
  {
    end = new Date();
    console.log(i, end - start);
    start = new Date();
  }
}
```

- Case 2) Using prisma (https://github.com/prisma/prisma/issues/10439)
  - Create levenshtein function



루틴 본문:

```
BEGIN
    DECLARE s1_len, s2_len, i, j, c, c_temp, cost INT;
    DECLARE s1_char CHAR;
    -- max strlen=255
    DECLARE cv0, cv1 VARBINARY(256);

    SET s1_len = CHAR_LENGTH(s1), s2_len = CHAR_LENGTH(s2), cv1 = 0x00, j = 1, i = 1, c = 0;

    IF s1 = s2 THEN
        RETURN 0;
    ELSEIF s1_len = 0 THEN
        RETURN s2_len;
    ELSEIF s2_len = 0 THEN
        RETURN s1_len;
    ELSE
        WHILE j <= s2_len DO
            SET cv1 = CONCAT(cv1, UNHEX(HEX(j))), j = j + 1;
        END WHILE;
        WHILE i <= s1_len DO
            SET s1_char = SUBSTRING(s1, i, 1), c = i, cv0 = UNHEX(HEX(i)), j = 1;
            WHILE j <= s2_len DO
                SET c = c + 1;
                IF s1_char = SUBSTRING(s2, j, 1) THEN
                    SET cost = 0; ELSE SET cost = 1;
                END IF;
                SET c_temp = CONV(HEX(SUBSTRING(cv1, j, 1)), 16, 10) + cost;
                IF c > c_temp THEN SET c = c_temp; END IF;
                SET c_temp = CONV(HEX(SUBSTRING(cv1, j+1, 1)), 16, 10) + 1;
                IF c > c_temp THEN
                    SET c = c_temp;
                END IF;
                SET cv0 = CONCAT(cv0, UNHEX(HEX(c))), j = j + 1;
            END WHILE;
            SET cv1 = cv0, i = i + 1;
        END WHILE;
    END IF;
    RETURN c;
END
```

  - Create code

```
//CASE2 : 3000개당 약 850ms 소요 (사용불가)

const result = await prisma.$queryRaw
`SELECT * FROM platenumber WHERE levenshtein(plate, ${plateNumber})
BETWEEN 1 AND 2;`
```

- Case 3) calculating Levenshtein(a, b) via npm library after pattern matching
    - Step1) Pattern matching
        - Find pattern matching using regex(regular expression)
        - If the first 3 letters of the requested license plate are not numbers but just alphabets, find all license plates that match the first 3 letters.

```
else //if no exact match
{
  var partial_result = [];
  let word, number;
  var result;

  //console.log(plateNumber.length)

  word = plateNumber.substr(0, 3);

  if (/^[a-zA-Z]+$/.test(word)) //앞에 3자리가 문자이면 partial match수행
  {
    const query = new RegExp('^'+ word);
    try {
      result = await collections.find(
        {
          plate: query
        }
        ).maxTime(FIND_TIME_OUT);
      } catch (err) {
        res.json(result);
        console.log(req.user.id+' Search Timeout');
        logfn( req.user.id, req.params.platenumber, start, new Date().getTime(), 'no');
      }
  }
```

    - Step 2) find levenshtein distance based on set minimum confidence after searching step 1).

```
if (result.length)
{
  for (var i=0;i<result.length;i++)
  {
    if (levenshtein.get(plateNumber, result[i].plate) <= dist_levenshtein)
    {
      partial_result.push(result[i]);
    }
  }
  if (partial_result.length > max_num_of_partial_match)
  {
    res.json(partial_result.slice(0,max_num_of_partial_match));
  }
  else
  {
    res.json(partial_result);
  }
  console.log(req.user.id+' partial match '+partial_result.length);
  logfn( req.user.id, req.params.platenumber, start, new Date().getTime(), 'partial');
}
```

- If the first 3 letters of the requested license plate include numbers or the no result of 1) and 2), then the matching result is "NO RESULT".

Duration

Experiment duration : 11 Jul ~ 15 Jul

Links and references

Distance de Levenshtein en mySQL (https://lucidar.me/fr/web-dev/levenshtein-distance-in-mysql/)

Experiment 7: Improve the accuracy

Results and recommendations

We made three versions of detection logic which have some different parameter values in the config files(openalpr.conf, us.conf), and they have almost the same accuracy. But when we applied the Maximum Likelihood Voting Technique that uses the confidence of the detected result of each version to make a more accurate estimation of the most likely correct result. Finally we could improve the accuracy by 3.7% compared to the original version of ALPR. In case of beaver3.avi, the accuracy is improved over 5%. Though the usage of CPU and Memory increased, it was an acceptable boundary.

Objective

- To improve the accuracy of the license plate number that ALPR detects
- we have quality attribute
  (QA-06) ALPR shall maximize recognition accuracy
  We will improve the accuracy for the studio project.

Status

[Planned | In progress | Suspended | Canceled | **Concluded**]

Expected outcomes

The recognition accuracy of ALPR should be improved by at least over 2 percent.

We recognized in many papers that it’s hard work to improve the accuracy by at least over 1 percent in the deep learning field.

Resources required

1. Test DB

   We used 4 avi files for the Test DB that was shared and did the annotation work on every license plate.

no	file name
1	beaver1.avi
2	beaver2.avi
3	beaver3.avi
4	beaver4.avi

2. Accuracy metric

   We used two metrics to measure the accuracy for Test DB. The number of correctly detected license plate numbers among existing license plate numbers.

$$Accuracy = \frac{\text{The number of rightly detected license plate numbers}}{\text{The number of existing license plate numbers}}$$

3. 1 person’s 8 days effort

Experiment description

Experiment procedure

Step 1. Calculate the accuracy in given sources with the model file and config file. It will be used to compare how much the accuracy is improved and one of the versions for Maximum Likelihood Voting Technique.

[Result of Step 1]

no	File name	Accuracy(%) of original(version1)
file1	beaver1.avi	94.28
file2	beaver2.avi	100
file3	beaver3.avi	82.35
file4	beaver4.avi	100
Total	**Total**	**90.12**

[The accuracy of original version(called version1)]

Step 2. Make the two more versions of detection logics for Maximum Likelihood Voting Technique.

[Result of Step 2]

1) **version2**
   - We've tuned the parameter **'char_height_mm**' in the us.conf file. Though the accuracy is not improved it could detect some license plate number that version 1 or version 3 couldn't detect

file (including parameter)	changed parameter	value before changed	value after changed
us.conf	char_height_mm	70	75

[The changed parameter list of version2]

2) version3

- We've tuned the parameters **'char_height_mm', 'plate_height_mm', 'min_plate_size_height_px'** in the us.conf file and **.'postprocess_min_confidence'** in the openalpr.conf file. Though the accuracy is not improved it could detect some license plate number that version 1 or version 3 couldn't detect

file (including parameter)	changed parameter	value before changed	value after changed
us.conf	char_height_mm	70	75
us.conf	plate_height_mm	152.4	100
us.conf	min_plate_size_ height_px	35	45
openalpr.conf	postprocess_min _confidence	65	80

[The changed parameter list of version3]

no	File name	Accuracy(%) of version2	Accuracy(%) of version3
file1	beaver1.avi	94.28%	91.42%
file2	beaver2.avi	100%	100%
file3	beaver3.avi	82.35%	85.29%
file4	beaver4.avi	100%	100%
Total	**Total**	**90.12%**	**90.12%**

[The accuracy of version2 and version3]

Step 3. Use Maximum Likelihood Voting Technique with three versions of detection logic. The voter uses the confidence of the detected result of each version and chooses the result that has the highest value of confidence to make a more accurate estimation of the most likely correct result. Then we could improve the accuracy by 3.7% compared to the original version of detection logic. In case of beaver3.avi, the accuracy is improved over 5%.

[Result of Step 3]

no	File name	Accuracy(%) (when applied Maximum Likelihood Voting Technique)
file1	beaver1.avi	97.14%(▲2.86%)
file2	beaver2.avi	100%(-)
file3	beaver3.avi	88.23%(▲5.88%)
file4	beaver4.avi	100%(-)
Total	**Total**	**93.82%**(▲3.7%)

[The accuracy when applied Maximum Likelihood Voting Technique]

Though the usage of CPU and Memory increased, it was an acceptable boundary. Below is the comparison of the usage of CPU and Memory between the original version and the last version which applied Maximum Likelihood Voting Technique.

	CPU (%)	Memory (MB)
original version	2~3	160~173
version that is applied Maximum Likelihood Voting Technique	4~5	320~340

Duration

Experiment duration : 29 June ~ 10 Jul

Links and references

1) https://www.openalpr.com/benchmarks : To determine the baseline of the accuracy.

2) https://tesseract-ocr.github.io/tessdoc/Data-Files.html : The recently updated Tesseract models

Architecture Decision Records(ADR)

ADR 1: Use Model-View-Controller(MVC) pattern

Only six weeks which is not enough time for the studio project is given and some requirements could be possibly changed further since we cannot figure all the requirements in the beginning due to the lack of the time. Therefore, the UI developer should be able to easily change and test the user interface within a few hours. [QA-01]

Decision

We will apply the MVC pattern for the UI developer to change the user interface properly in a short amount of time.

Rationale

Model-View-Controller(MVC) pattern promotes clear separation of concerns on the layout of the UI and improves the **modifiability** quality attribute. Although It has tradeoffs that increase the complexity and a small amount of latency to user interactions which is short enough to be ignored, its benefit is so powerful in terms of the fast update of the user interface.

Furthermore, we conducted Experiment 3: Streaming performance test for UI and ALPR regarding how to share the video stream with both the ALPR engine and the UI. We considered potential designs such as MVC and Publish-subscribe patterns.

Rejected alternatives and rationale as follows:

1. Publish-subscribe pattern: the UI and the rest of the modules can be independent using the publish-subscribe pattern. However, it can negatively impact latency due to too many messages, also causing difficult debugging.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- The UI developer can easily change and be focused on the UI.

ADR 2: Use React Front-end

The client must provide the UI showing the information of the license plate found, the preview, the alert and so on [FR-03/FR-04/FR-05/FR-06/FR-07/FR-11].

There are various UI frameworks in which the programming languages are different from each other. Therefore, we have to consider which programming languages are preferred by the developers in our project due to the learning curves of the programming languages.

Regarding *OpenALPR*, the ALPR engine we use in our project, is developed by C++, no one is familiar with C++ UI framework such as MFC in our project. Furthermore, even if we use MFC, it is also old fashioned while there are trendy and stylish UI frameworks out there.

Decision

We will use **React front-end** which is a trendy and popular UI framework based on Javascript, easy to learn. Furthermore, it is recommended to launch the application on Chrome kiosk mode since React is running on the browser.

Rationale

In general, React, a javascript library for building user interfaces, is a popular UI framework with a huge amount of open libraries called node packages. It is easy to develop since it is based on Node.js (Javascript) and provides declarative UI programming and component-based logic. Furthermore, it is trendy and stylish with the UI libraries constantly updated with the latest trend.

Also specifically, we have a developer who has been developing React applications for years, and we found a UI template under MIT license so that we can use it in our project.

Rejected alternatives and rationale as follows:

- *MFC or WPF/Winform*: It would be efficient if the UI is implemented by MFC so that it communicates with the ALPR engine by function calls locally. However, MFC is complicated to develop causing a learning curve and is old-fashioned in terms of the UI.
- *QT UI*: QT is a well-known cross-platform UI framework and easier than C++. However, how to communicate between Python and C++ ALPR is not figured out at the moment.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- React front-end is added to the system.
- A React UI template under MIT license is applied to the system.The communication between React UI and the C++ ALPR engine is newly introduced and must be considered. (The complexity of the system has increased.)
- The system can provide a stylish and trendy UI that comes with famous UI libraries such as *Material-UI*, *Bootstrap* and so on.
- We can easily develop the UI and furthermore, we can use various open libraries for the client as well as the UI.
- As a tradeoff, we have to figure out how to share the video stream for both ALPR detection and UI preview.

ADR 3: Use Pipe-and-Filter and manage sampling rate tactic for UI preview

The UI and the ALPR engine are separated in our system and the video stream is sent via the websocket. Therefore the quality of the preview depends on the websocket network traffic, and the user should be able to change the rate of the sending ALPR preview image (fps) so the preview is displayed with the set rate once the user adjusted. [QA-07]

Decision

We will apply the pipe-and-filter style and the *manage sampling rate* tactic for the flow control of the video stream from the ALPR engine to the UI preview.

Rationale

We conducted Experiment 4: Communication method test for UI and ALPR to find an appropriate communication method for the video stream between the UI and the ALPR engine, and we decided to use websocket. For that, we can do the flow control at runtime.

Applying the pipe-and-filter style, the rate of the sending of the video stream can be changed **at runtime**. Also using *the manage sampling rate* tactic, which is one of the performance tactics, can be used to reduce the frequency of the sending images to the UI for preview when the device has the low CPU performance. The lower the CPU performance, the lower the preview quality.

Furthermore, it is hard to adjust the rate automatically because the circumstances keep changing. Therefore, it is reasonable to let the user set the rate by themselves.

Rejected alternatives and rationale as follows:

1. *Manage event arrival tactic*: There is an alternative that the ALPR engine can send the image sequence as best as it can and then the UI preview consumes with best effort. However, it is inefficient because it causes so much network traffic.
2. *Increase resources tactic*: Using the GPU is *Deus Ex Machina*, the most powerful solution. However, the GPU is not always guaranteed in the client.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- The system can provide a stable UI preview by setting *fps* flexibly.
- The UI menu to set *fps* is added.

ADR 4: Use Websocket with ping/echo tactic for network failure

The client should detect network connectivity issues with the backend system within a few seconds and automatically resolve the communication issue (QA-02).

Decision

We will use **websocket** which detects network connectivity issues based on polling(ping/echo tactic) within 3 seconds and also tries to reconnect for every N second.

Rationale

Websocket provides various event listeners related to network connectivity issues based on the ping/echo mechanism. It invokes the callback function *onclose()* upon detection of network failure, so we can try to reconnect by invoking *connect()* after N seconds, where N is pre-defined or given by the user. In this way, the network will be automatically recovered as soon as it becomes available again.

Rejected alternatives and rationale as follows:

- Heartbeat tactic: A server can send a heartbeat message periodically if it sends a response regularly. However, the request of the client is not regular.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- New websocket connection is added with an option that tries to reconnect for every 1 second once disconnected.
- Alert UI is connected with the event listeners of the websocket.

ADR 5: Use C++ openALPR as the license plate recognition engine

To achieve the quality attribute that ALPR shall maximize recognition accuracy [QA-06], we considered C++ OpenALPR and Rekor Scout Commercial ALPR.

Decision

We decided to use C++ OpenALPR that provides the ALPR engine freely. We don't have a budget for the commercial ALPR.
And improved the accuracy of C++ OpenALPR.(TE#7)

Rationale

We considered both C++ OpenALPR and Rekor Scout Commercial ALPR as the license plate recognition engine. C++ OpenALPR is free for use but there is a charge to use the Rekor Scout Commercial ALPR.
Because we don't have the budget for the charge to use the Rekor Scout Commercial ALPR, we chose the C++ OpenALPR as the license plate recognition engine though Rekor Scout Commercial ALPR has better performance in accuracy.
For accuracy, we improved the precision by 3.7%. Regarding improving the accuracy, see the technical experiment(TE#7).

Rejected alternatives and rationale as follows:

1. Rekor Scout Commercial ALPR : If we use Rekor Scout Commercial ALPR for the license plate recognition engine, we can achieve high accuracy.
   But we don't have budget for the charge to use the Rekor Scout Commercial ALPR, we choose the C++ OpenALPR as the license plate recognition engine

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- We can use the license plate recognition engine free of charge for the use.
- To achieve enough accuracy, improved the accuracy by technical experiment(TE#7)

ADR 6: Use Facade pattern on C++ openALPR

C++ ALPR has a lot of features like

- C++ Main program
- Communication channel to React Front-end
- ALPR precondition
- ALPR state control (start / stop)
- ALPR frame control when it is congested so it doesn't guarantee 25 fps[QA-05].

Decision

We decided to apply a facade pattern on C++ ALPR, so we extract ALPR features from LGDemo to ALPRFacade. so it can execute ALPR processing easily when ReactJS Front-end requests to run ALPR.

Rationale

We're not familiar with all the ALPR libraries and this library has a lot of features and usage patterns.. and we think that it is really error-prone when it doesn't have fixed usage patterns.

And we have to refactor LGDemo because it has all in the main function, it can't be used for any external input. so we amoves ALPR features to ALPRFacade, it limits ALPR usage patterns but it offers verified ALPR functions to external actor like ReactJS

Rejected alternatives and rationale as follows:

1. Monolithic LGDemo : It contains all the features only on C++ main function.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- Analyze LGDemo features.
- Design interface of ALPRFacade.
- Implement ALPRFacade
- Verify ALPRFacade interface is compact and well equipped by testing with ReactJS Front-end.

ADR 7: Use Node.js Express Server

The server must provide the multi-client connections and the database connection [FR-08/FR-09/FR-10/FR-12]. Furthermore the server also should provide secured and authorized communications based on user authentication [CST-07/FR-24/FR-25].

Although there are well known server frameworks, each framework uses a different programming language so it depends on which framework is familiar to our developers.

Decision

We will use **Node.js Express Server** which comes with various libraries.

Rationale

In general, Node.js express, a web application framework, is a popular server framework with a huge amount of open libraries called node packages. It is easy to develop since it is based on javascript and provides secured communications using libraries such as TLS/HTTPS, JWT, etc. Furthermore, it is easy to connect many different types of databases.

Also specifically, we have a developer who has been developing Node.js express server for years.

Rejected alternatives and rationale as follows:

- *Spring framework*: It is also a widely used framework and a big community where we can get help. However, it is somewhat heavy and complicated since it is based on Java and Maven.
- *Python Django*: It is also a well-known framework and lightweight. However, our developers are more familiar with Javascript than Python.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- Node.js Express Server is added to the system.
- Node packages will be used for server features such as encryption, authentication, and so on.

ADR 8: Use Chaining middleware for server's RESTful APIs

After that we chose node.js and express as a server providing RESTful APIs. We have to design how to implement collateral features like authentication[FR-24,FR-25] and performance logging[FR-10].

Decision

We decided to apply chaining middleware for server's RESTful APIs. Doing so we can easily add additional common features to RESTful API gateway. and we can add and remove collateral features only on entry we want.

Rationale

As-is :

```
    app.get('/platenumber/:platenumber', (req, res) => {
      return apiPlate(req, res, addPerfLog)
    });
```

To add some auxiliary features, we have to modify the functions directly. so we end up adding coupling on the function.

To-be:

We make each auxiliary feature as separate middleware for express on node.js.

```
    const authenticateAccessToken = (req, res, next) => {
      let authHeader = req.headers["authorization"];
      let token = authHeader && authHeader.split(" ")[1];

      if (!token) {
          console.log("wrong token format or token is not sended");
          return res.sendStatus(400);
      }

       jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (error, user)
=> {
```

```
            if (error) {
                console.log(error);
                return res.sendStatus(403);
            }

            req.user = user;
            next();
        });
    };
```

```
    const addPerformanceLog = (req, res, next) => {
      next();
      addPerfLog(  req.user.id,  req.platenumber  ,  res.starttime,  new
Date().getTime(), res.match_type );
    };
```

After that, we can easily add any auxiliary functions to the RESTful API we want. and we can use chaining functions like below when we want to multiply auxiliary functions.

```
app.get('/platenumber/:platenumber',authenticateAccessToken,  addPerfLog
(req, res) => {
      return apiPlate(req, res )
  });
```

Rejected alternatives and rationale as follows:

1. Add auxiliary functions directly on REST api : It has coupling and modifiability problems by containing supporting features.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- Analyze Node.js + Express's RESTful API support.
- Design middleware we need to meet our requirements.
- Implement each middleware function.
- Apply chaining middleware to the RESTful API we want.

ADR 9: Use detect intrusion, authenticate & authorize users and inform actor tactics to improve security on RESTful communication between client and server

The interface between the client and the server is required and it should be secured and [CST-07] only authorized users should be able to communicate with the server to query car information and the server should provide user authentication. [FR-24/FR-25]
Furthermore, the system should detect an illegitimate access and notify the administrator immediately when an unauthorized user attempts to access the system API. [QA-03]

Decision

We will apply the **Detection Intrusion** tactic to prevent our system from hackers' attack. and we will use JWT(Json Web Token) to **authenticate and authorize users**. so we can limit restricted resources from unauthorized access. MQTT was applied to **notify administrators** when unauthorized access is detected.

Rationale

The Detect Intrusion tactic compares service request patterns within a system to a set of signatures or known patterns of malicious behavior. In our project, the malicious behavior can be determined by checking the authorization token once the user is authenticated.

Prior to applying JWT, we conducted Experiment 5: Install X.509 Certificate to check if it works properly on HTTPS, the secure http session, to reduce the security threat.
so.
To authenticate and authorize users, we can use JWT. by using JWT we can avoid repeating a query to the authorization server to verify users or tokens.
Based on HTTPS and JWT, we choose RESTful APIs as communication methods. By using RESTful APIs, we can easily make and debug messages between client

and servers. otherwise we have to make additional message dispatchers or proxies / stub for remote communications.

And when it senses abnormal unauthorized access like
- 5 continuous access attempts without authorization within 10 seconds.

System will notify immediately to the administrators.

Rejected alternatives and rationale as follows:

1. Raw TLS / SSL : It has a lot of things to care about. We have to make raw secure communication channels, and have to take care of messages flow control, message dispatch and so forth.
2. Token authentication(Not JWT): It can avoid repeating password exposure, but can't avoid repeating queries to the authorization server (It's just a DB query in this project).
3. gRPC : It is harder than RESTful API to debug rather than. because it is based on binary messages. and we're not familiar with gRPC at the moment.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- Apply self-signed certificate on node.js server for HTTPS, and found out it can offer a secure channel as we expect. [TE#5]
- Design and implement JWT libraries for the server.
- Design and Implement RESTful APIs between client and server.
- Make intruder detection rules to judge whether it's reasonable unauthorized access or it seems like attacks from external hackers.

ADR 10: Use MongoDB for searching license plate

“The system should respond to plate number detection requests within 1 second when DB has 25,000,000 records for car detection.” (QA-04)

When the system query the car information in view. We should alarm or notify the car information to the user while the car is still in view.

So we investigate all the video footage provided for the studio project. and we found out all the plates stay in the view for at least 1 second. and the driver drives the car at a low speed to examine cars. so we think that 1 second is enough to process querying license plate

We compare SQL and NoSQL databases.

Decision

We will use MongoDB (NoSQL) as a database to store records about vehicles. Because MongoDB has better performance in READ operations, and Most of our user scenarios depend on READ operations, so we will use MongoDB.

Rationale

MongoDB improves **performance** quality attributes for exact matches. We tested MariaDB and MongoDB and both DBs can handle 25M records and query exact matches. In addition, Modifiability could be similarly applied to both MongoDB and MariaDB. But exact matching using MongoDB performs faster than MariaDB. MongoDB performs within 10ms for exact match. On the other hand, MariaDB performs within 40ms.

Rejected alternatives and rationale as follows:

MariaDB : MariaDB, like MongoDB, can handle 25M records and has excellent response time for exact matching. However, the performance is inferior compared to MongoDB.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- The system can provide excellent performance time for exact matching. (TE#1)
- Set MongoDB as server's plate number DB.
- Insert index on plate number field.
- Install package supporting levenstein distance.
- Monitor round trip time for query plate number from client to server.

ADR 11: Use Levenshtein Distance after pattern matching for Partial Match

The system should return the best match license plate if there is not an exact match that includes a configurable minimum confidence threshold to support a partial match. [FR-22]

The system should respond to plate number detection requests within 1 second when DB has 25,000,000 records for car detection.” (QA-04)

Decision

We decided to use Calculating Levenshtein after pattern matching(Case 2) as mentioned in TE#6.
Applying Case 2, we improved performance using **Increase efficiency of resource usage** tactics. Also, we used **Bound execution times** tactics to prevent the detection time over 1 second.

Rationale

After TE#6, we get the following test report from the table below.

Case	Description	pros	cons
Case 1	Just calculating Levenshtein for each plate in DB	Operating partial match is possible based on levenshtein distance for all plate numbers in DB	Processing time is too long (Approximately 2.5s per 1000 records)
Case 2	Calculating Levenshtein via npm library after pattern matching	processing time is greatly reduced and it solves the processing time in Case1. **(Average Processing Time : 150ms)**	The use of pattern matching has a constraint. (Constraint: The first 3 letters must be fixed)

As the above experimental results, calculating levenshtein after pattern matching could solve the problem of searching time.

Rejected alternatives and rationale as follows:

Case1 : Calculating Levenshtein distance for full searching in DB is possible, but searching time is too long.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- Calculating Levenshtein distance for full searching in DB is too long a processing time.
- We were able to reduce the searching time by using pattern match.
- But there is a constraint that reasonable time is measured only when the first 3 letters are fixed.

ADR 12: Use X509 Self Signed Certificate

The server must use a certificate to achieve TLS based secure communication. If the location does not provide a public IP address for the server, then it would not be possible to use an official certificate. We will test if we can get an official certificate at our location. The test will be done with Experimental #5. If it is not possible, then we will set the constraints [CST-06], a user *must proceed to the unsafe certificate at the very beginning.*

Decision

We will use **X509 Self Signed Certificate.**

Rationale

The WeWork office does not provide a public IP address for users. While making a certificate, **Letsencrypt** requires a public IP address. So without the public IP address, we should use X509 self signed certificate. We will generate and use the certificate from OpenSSL and install it to the server. However this causes a user to agree to the security warning on the browser by clicking the button at the very beginning, connecting https://server_ip_address:3503/

Rejected alternatives and rationale as follows:

- *Official Certificate*: Without a public IP address, it is impossible.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- Node.js Express Server is added to the system.
- X509 Self signed service will be used by the server for TLS connection.

ADR 13: Use Maximum Likelihood Voting Technique to improve the accuracy

To achieve the quality attribute that ALPR shall maximize recognition accuracy [QA-06], we considered using Maximum Likelihood Voting Technique and updated OCR engine by training with the collected train data.

Decision

We decided to use Maximum Likelihood Voting Technique that uses the confidence of the detected result of each version to make a more accurate estimation of the most likely correct result.

Rationale

We recognized some difference in detected results when tuning the parameter, but the total accuracy was almost the same every tuning time. And we could also see, though one version of detection logic in which some parameters are tuned cannot detect a specific license plate number, the other version in which the different parameters are tuned could detect that license plate number.

For such reasons, we made three versions of detection logic which have some different parameter values in the config files(openalpr.conf, us.conf) and applied the Maximum Likelihood Voting Technique that uses the confidence of the detected result of each version and chooses the result that has the highest value of confidence to make a more accurate estimation of the most likely correct result.

Finally we could improve the accuracy by 3.7% compared to the original version of ALPR.

The accuracy may be certainly improved by training the train data(collected license plate number) and making a new OCR engine. But we don't have enough time to collect the license plate number and train them with tuning the model.

Because we don't have enough time to make a new OCR engine, we chose the Maximum Likelihood Voting Technique. And though the usage of CPU and Memory increased, it was an acceptable boundary. Regarding improving the accuracy and the usage of CPU and Memory, see the technical experiment(TE#7).

Rejected alternatives and rationale as follows:

1. Updating OCR engine by training with the collected train data : The accuracy may be certainly improved by training the train data(collected license plate number) and making a new OCR engine. But we don't have

enough time to collect the license plate number and train them with tuning the model.

Status

[Proposed | **Accepted** | Deprecated | Superseded]

Consequences

- We can improve the accuracy within a limited time.
- Though the usage of CPU and Memory increased, it was an acceptable boundary.
- To achieve enough accuracy, improved the accuracy by technical experiment(TE#7)