ARCHITECTURE DOCUMENT

C4 architecture

Enigma

GROUP MEMBERS:

BRIAN ONUOHA

LUC MOONEN

MICHAEL EBOWUSIM

RICK VAN HAM

CRISTIN RUSNAC

Version control

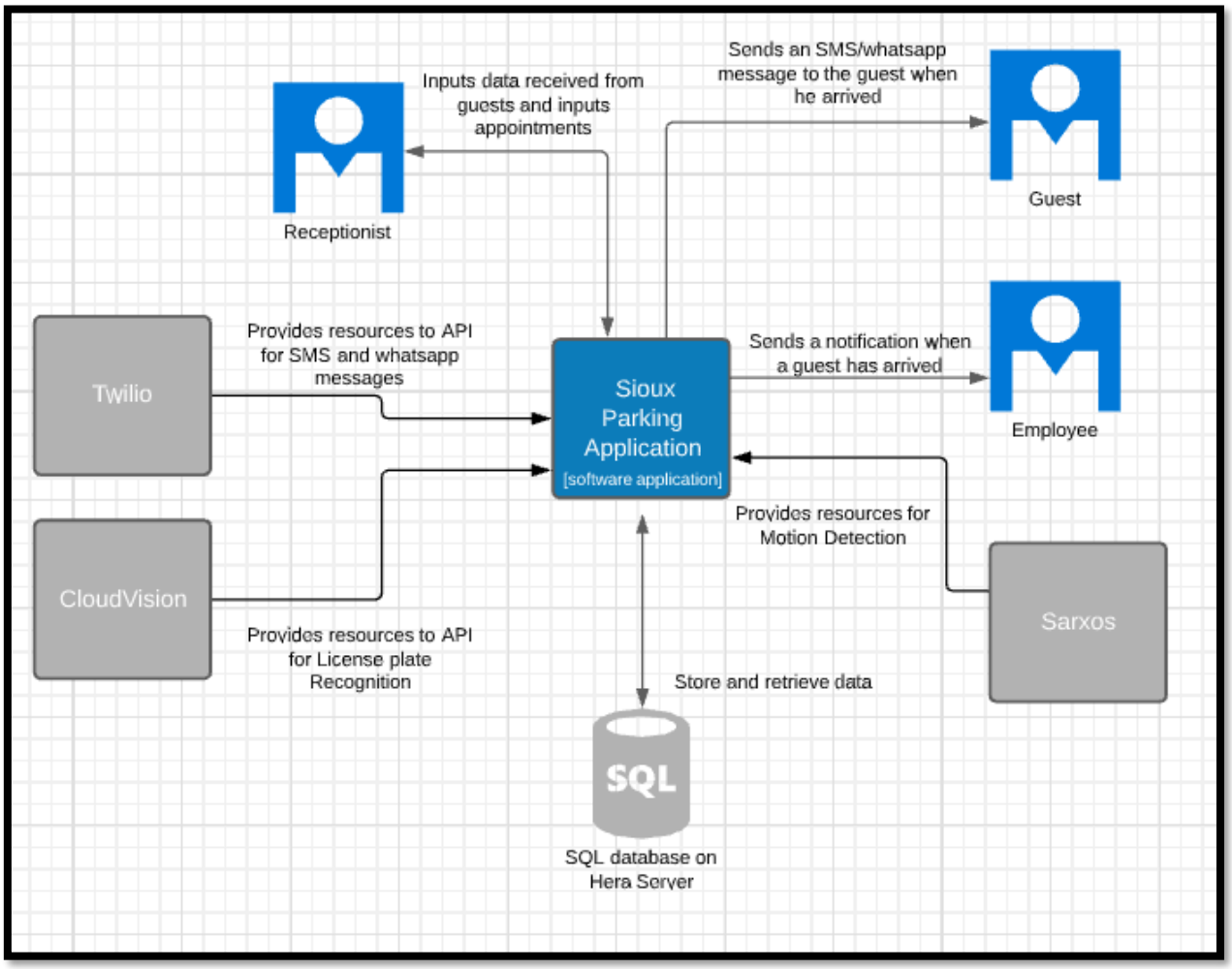
- **Version 1: (4/16/2021) Edited by Rick van Ham**
  Added Architecture C4 model.
- **Version 2: (5/16/2021) Edited by Rick van Ham**

Improved criteria received from teachers, added small explanation per C, added version control.

- **Version 3: (6/9/2021) Edited by Rick van Ham**
  Updated document to be compliant with the current project.
- **Version 4: (6/23/2021) Edited by Rick van Ham**
  Updated document to be compliant with the current project.

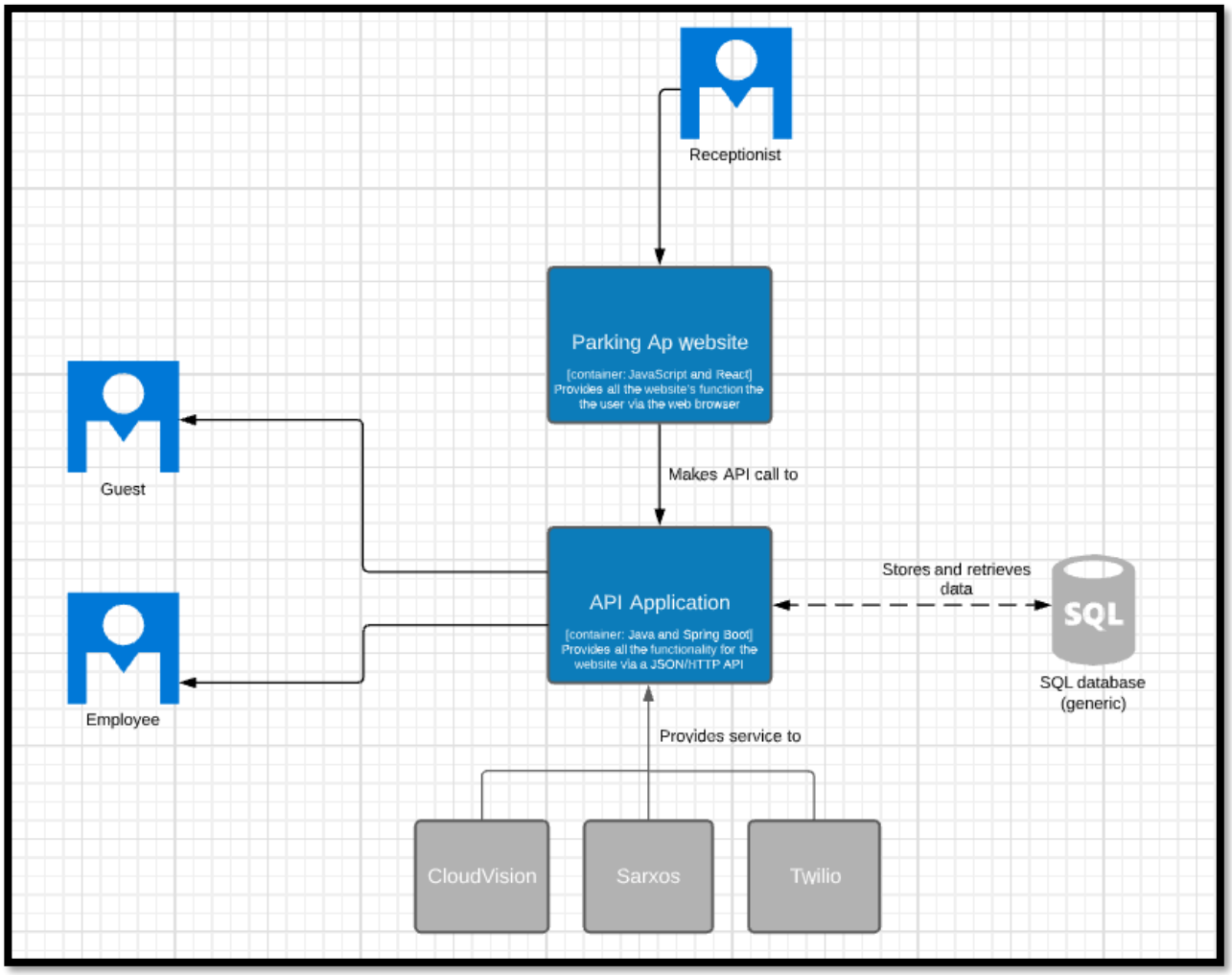
Contents

System Context

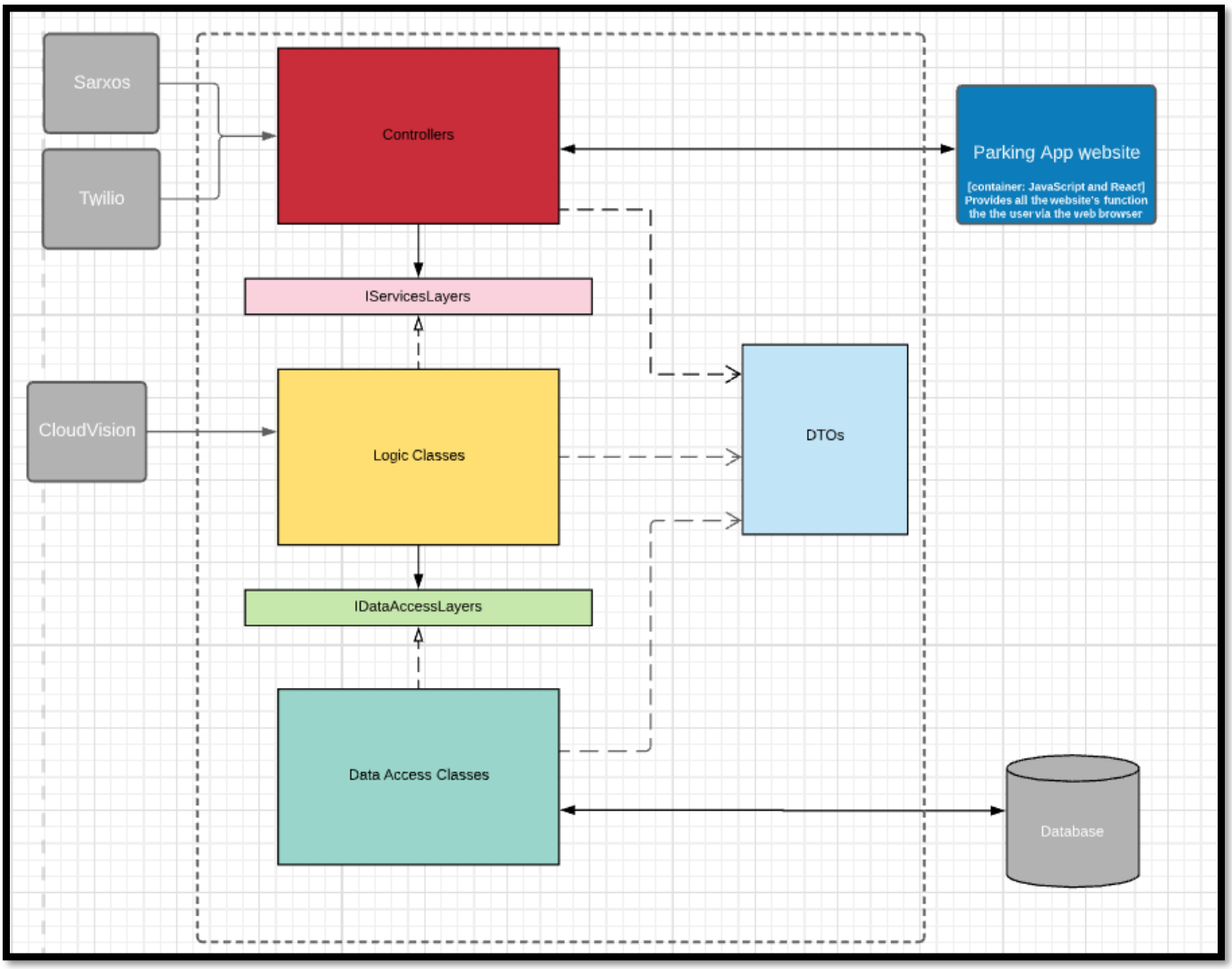


In this diagram, you see 3 different “Users”. The Receptionist who will use the application by inputting data into the application, as well as adding appointments, the Guest who will receive an SMS/WhatsApp message when he arrives, and the employee who will receive a notification (Being a notification on the desktop or an email) when the guest has arrived. External systems are seen as well. We have a Hera DB server that is used to store and retrieve data. We also have external APIs such as Twillio (Used for SMS/Whatsapp), CloudVision (Used for License plate Recognition) and Sarxos (For the motion detection). We use them in our API for these actions, but it’s the API that’s doing the actions, so that’s why the API sends an SMS to the guest and not Twillio.

C2 CONTAINERS AND TECHNOLOGY

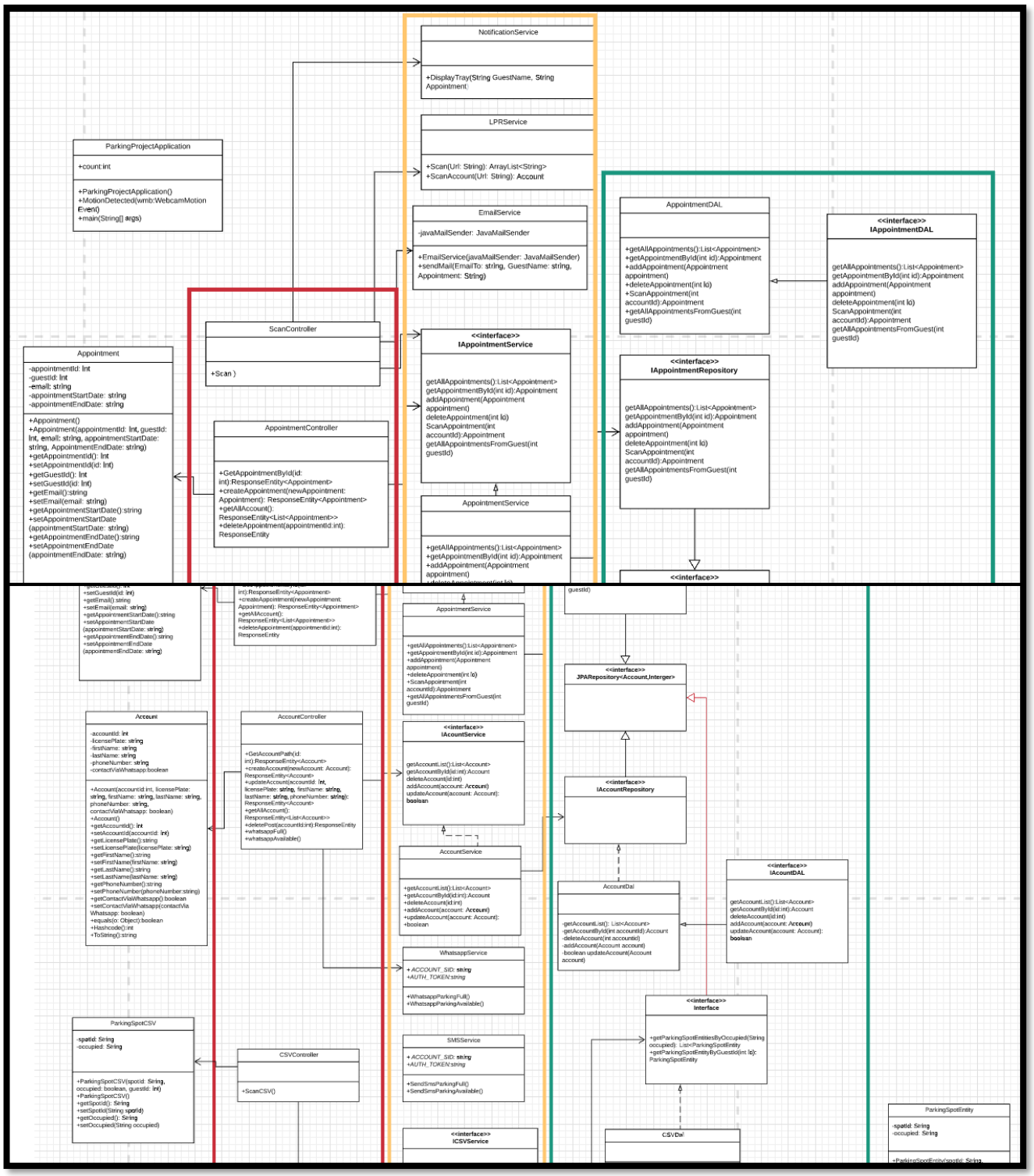


This diagram is a zoomed-in version of C1. Our application consists of 2 containers. The website made using React framework and Javascript and the API application made using Java and Springboot. The Receptionist visits the website where he inputs data from the client. The website makes an API call to the API who would store/ retrieve the requested data via the External Hera DB. The guest and the employee will not interact with the website, since they won't be needing to.



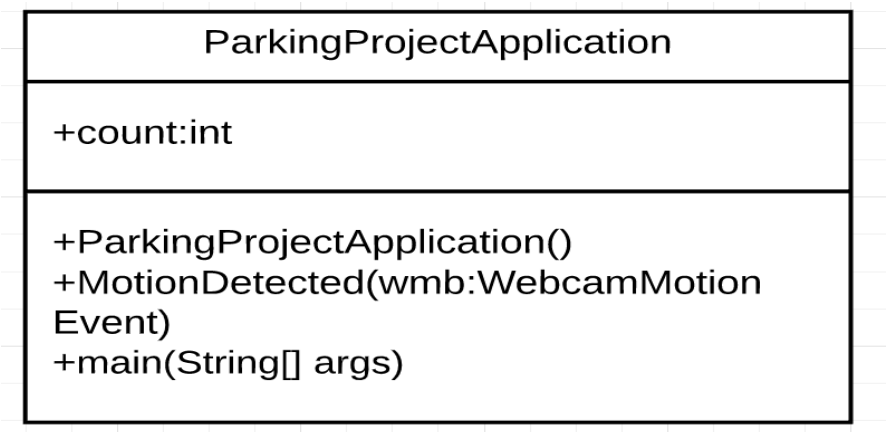
In this diagram, it's more zoomed-in into our API application. Our classes are divided into certain layers for a better-optimized flexible application with follows the SOLID principles. Controllers are the ones that will receive and return the request call from the front-end. Logic classes are the ones that will do the logic and the algorithm stuff. Data access classes are the ones that will retrieve/store data to the external database. They are the only layer that has access to the database. To send the data to each layer, they use DTOs (Data transferable objects) that contain the data (Guest, appointment). The logic classes and Data access classes inherit from an interface layer. This is done for Dependency Inversion (Solid) to make it more flexible. The CloudVision is used in the Logic class, while Twilio is used in the Controller class. Sarxos is actually used in the main class, because we wanted the webcam feature to work as soon as the application starts. The reason for this is unknown, but probably because the SMS feature was used by the controller as well.

C4 Classes


NotificationService
+DisplayTray(String GuestName, String Appointment)
LPRService
+Scan(Url: String): ArrayList<String>
+ScanAccount(Url: String): Account
ParkingProjectApplication
+count:int
+ParkingProjectApplication()
+MotionDetected(wmb:WebcamMotionEvent)
+main(String[] args)
EmailService
-javaMailSender: JavaMailSender
+EmailService(javaMailSender: JavaMailSender)
+sendMail(EmailTo: string, GuestName: string, Appointment: String)
AppointmentDAL
+getAllAppointments():List<Appointment>
+getAppointmentById(int id):Appointment
+addAppointment(Appointment appointment)
+deleteAppointment(int Id)
+ScanAppointment(int accountId):Appointment
+getAllAppointmentsFromGuest(int guestId)
<<interface>>
IAppointmentDAL
getAllAppointments():List<Appointment>
getAppointmentById(int id):Appointment
addAppointment(Appointment appointment)
deleteAppointment(int Id)
ScanAppointment(int accountId):Appointment
getAllAppointmentsFromGuest(int guestId)
ScanController
+Scan)
<<interface>>
IAppointmentService
getAllAppointments():List<Appointment>
getAppointmentById(int id):Appointment
addAppointment(Appointment appointment)
deleteAppointment(int Id)
ScanAppointment(int accountId):Appointment
getAllAppointmentsFromGuest(int guestId)
<<interface>>
IAppointmentRepository
getAllAppointments():List<Appointment>
getAppointmentById(int id):Appointment
addAppointment(Appointment appointment)
deleteAppointment(int Id)
ScanAppointment(int accountId):Appointment
getAllAppointmentsFromGuest(int guestId)
Appointment
-appointmentId: Int
-guestId: Int
-email: string
-appointmentStartDate: string
-appointmentEndDate: string
+Appointment()
+Appointment(appointmentId: Int, guestId: Int, email: string, appointmentStartDate: string, AppointmentEndDate: string)
+getAppointmentId(): Int
+setAppointmentId(id: Int)
+getGuestId(): Int
+setGuestId(id: Int)
+getEmail():string
+setEmail(email: string)
+getAppointmentStartDate():string
+setAppointmentStartDate (appointmentStartDate: string)
+getAppointmentEndDate():string
+setAppointmentEndDate (appointmentEndDate: string)
AppointmentController
+GetAppointmentById(id: int):ResponseEntity<Appointment>
+createAppointment(newAppointment: Appointment): ResponseEntity<Appointment>
+getAllAccount(): ResponseEntity<List<Appointment>>
+deleteAppointment(appointmentId:int): ResponseEntity
AppointmentService
+getAllAppointments():List<Appointment>
+getAppointmentById(int id):Appointment
+addAppointment(Appointment appointment)
<<interface>>
JPARepository<Account,Interger>
<<interface>>
IAccountRepository
Account
-accountId: Int
-licensePlate: string
-firstName: string
-lastName: string
-phoneNumber: string
-contactViaWhatsapp:boolean
+Account(accountId:int, licensePlate: string, firstName: string, lastName: string, phoneNumber: string, contactViaWhatsapp: boolean)
+Account()
+getAccountId(): Int
+setAccountId(accountId: Int)
+getLicensePlate():string
+setLicensePlate(licensePlate: string)
+getFirstName():string
+setFirstName(firstName: string)
+getLastName():string
+setLastName(lastName: string)
+getPhoneNumber():string
+setPhoneNumber(phoneNumber:string)
+getContactViaWhatsapp():boolean
+setContactViaWhatsapp(contactViaWhatsapp: boolean)
+equals(o: Object):boolean
+Hashcode():int
+ToString():string
AccountController
+GetAccountPath(id: int):ResponseEntity<Account>
+createAccount(newAccount: Account): ResponseEntity<Account>
+updateAccount(accountId: Int, licensePlate: string, firstName: string, lastName: string, phoneNumber: string): ResponseEntity<Account>
+getAllAccount(): ResponseEntity<List<Account>>
+deletePost(accountId:int):ResponseEntity
+whatsappFull()
+whatsappAvailable()
<<interface>>
IAccountService
getAccountList():List<Account>
getAccountById(id:int):Account
deleteAccount(id:int)
addAccount(account: Account)
updateAccount(account: Account): boolean
AccountService
+getAccountList():List<Account>
+getAccountById(id:int):Account
+deleteAccount(id:int)
+addAccount(account: Account)
+updateAccount(account: Account): +boolean
AccountDal
-getAccountList(): List<Account>
-getAccountById(int accountId):Account
-deleteAccount(int accountId)
-addAccount(Account account)
-boolean updateAccount(Account account)
<<interface>>
IAcountDAL
getAccountList():List<Account>
getAccountById(id:int):Account
deleteAccount(id:int)
addAccount(account: Account)
updateAccount(account: Account): boolean
WhatsappService
+ ACCOUNT_SID: string
+AUTH_TOKEN:string
+WhatsappParkingFull()
+WhatsappParkingAvailable()
SMSService
+ ACCOUNT_SID: string
+AUTH_TOKEN:string
+SendSmsParkingFull()
+SendSmsParkingAvailable()
ParkingSpotCSV
-spotId: String
-occupied: String
+ParkingSpotCSV(spotId: String, occupied: boolean, guestId: Int)
+ParkingSpotCSV()
+getSpotId(): String
+setSpotId(String spotId)
+getOccupied(): String
+setOccupied(String occupied)
CSVController
+ScanCSV()
<<interface>>
ICSVService
<<interface>>
Interface
+getParkingSpotEntitiesByOccupied(String occupied): List<ParkingSpotEntity
+getParkingSpotEntityByGuestId(int Id): ParkingSpotEntity
CSVDal
ParkingSpotEntity
-spotId: String
-occupied: String
+ParkingSpotEntity(spotId: String,

Right here is a UML diagram of our API application. We apologize for the bad quality, so we've provided you with the link to the diagram.
https://lucid.app/lucidchart/invitations/accept/inv_c80c6141-1aaa-4eb3-bdce-b53903c00d70?viewport_loc=-2300%2C162%2C14331%2C6538%2C0_0
In Appointment, the start- and end dates are both String instead of Datetime. The reasoning for this is because in the front-end it wouldn't cause any errors if it was String. Also, Maikel, who worked on this project previously said it's better to make it string instead of Datetime.
Scan Controller uses EmailService, LPRService, NotificationService, and IAccountService. This is because we used ScanController to show the perfect scan (where the license plate gets recognized and everything works).
AppointmentService has no Update function, because of the JPA repository. JPA has a function called save(), which works both as Create and Update. So when we call AddAppointment(), it will check if that appointment is in the database. If it is, it will update it. If not, it will add to the database.

Account has a Boolean called contactViaWhatsapp that checks if the client wants to be contacted via WhatsApp or not. If it's not checked, the guest will receive a message via SMS.
ScanController will check if their spot is available for the client. If there is a spot available, it will send the message that the spot is available. If not, it will send the message that redirects the guest to the nearest parking space. Then it will check if that guest wants to be contacted via Whatsapp or not.
For storing the parking spots, the application reads a .csv file and it stores in the ParkingSpotCSV DTO. Due to a miscommunication with the client, we created a separate Entity class that is only used by the Data Access Layer called ParkingSpotEntity. When transferred from service to Data, the information from the DTO will be transferred to the Entity and vise versa. Since this method is also more safe since we're not using the same DTO for data access, we kept this way.



For the motion detection, there is a method in the root class called MotionDetected() that will take a screenshot and wait for 5 seconds every time it detects motion. This screenshot will be used in ScanController to see if it can recognize the license plate.