ACME Architectural Plan

Oliver Matthew Bowker (220263618)

August 25, 2023

Contents

List of Figures

List of Tables

1 Introduction

In this report I will expand on the work done previously for the company known as ACME. A system has been designed for the company to move into a digital and online world. This system was designed around the following business goals:

- Increase profits/customers
- Improve documentation resilience and navigability
- Cater to the student demographic
- Automate/speed up time intensive tasks

This new system was designed to move ACME away from a paper-based system, which was slow to update and had little to no *'backup-ability'*, as well as from using phone/email to communicate. In addition to this they wanted to take advantage of the student population. Research in previous work concluded that adding cryptocurrency payments could be a good way to engage this younger audience and could provide an interesting angle for marketing. The new software system would move all the old functionality into a software-based approach, these features include:

- Adding customer information
- Taking payment
- Handling the return of a vehicle
- Starting a new hire
- Adding a new car to the system

A web-based application is to be created to support the above features. Where users can book and manage accounts and rentals without the need of a member of staff. This web application will handle the payments, customer details, car details and rental details using databases and third-party providers that are discussed in the previous work.

This report will continue this work by looking at the project in terms of architectural design, any security and safety concerns that could arise from the project and how the software will handle faults and promote availability. I have included the sequence diagrams from the previous work in **Appendix A** to help understand the flow of the new systems.

2 Architecture of the new system

System architecture is described by Ian Sommerville as:

> *'Software architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.'* [1]

Following will be the proposed design for ACME and a discussion on programming languages and how they relate to design decisions made. Details of the design can be found in **Appendix B**.

2.1 Overall Architecture

The architecture I have chosen is a Cloud-Native architecture. AWS, the largest cloud provider [2], describes this architecture as the *'approach of building, deploying, and managing modern applications in cloud computing environments'* [3]. In simpler terms, developers can setup *'virtual'* servers that a third-party houses to run their software, they provide the infrastructure, you provide the code/instructions. These servers use virtualisation *'allows the hardware elements of a single computer ... to be divided into multiple virtual computers'* [4].

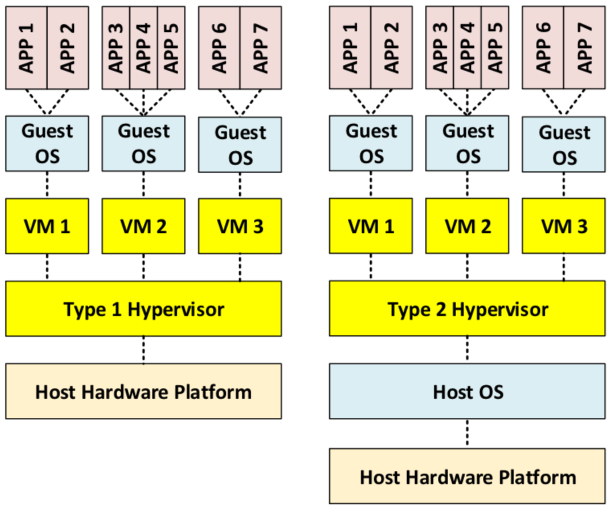


Figure 1: Diagram illustrating how virtualisation works [5].

2.2 Justification for a cloud-native approach

ACMEs' plan is an ambitious one, going from a paper-based system to a fully digitised solution that incorporates cryptocurrency are opposites! A large factor is ACME's lack of starting infrastructure. The purchasing of servers, database and account management software alone would cost a lot of money. This financial burden is somewhat lessened by using a cloud-native approach as you pay for what you use and companies such as AWS offer a free tier [6].

This type of architecture also enables the use of the sub-architectures like microservices which describe a *'single application [that] is composed of many loosely coupled and independently deployable smaller components' [7].*

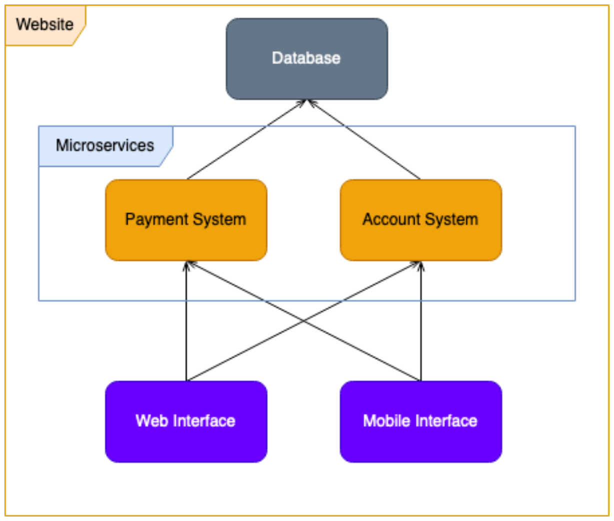


Figure 2: Simple diagram illustrating how a system can use microservices.

Customers will go to competitors if they deem your service to be unreliable or cannot access you're system. This is another area where cloud-native shines as they provide redundancy. AWS calls these AZs (Availability Zones) [8] these represent different data centers. So if one data center has an issue, your entire infrastructure can be *'ported'* to another one automatically. Coupled with this is the fact the services offered by these cloud providers have been tested by millions of people, so are resilient, but also the cost to develop some of these solutions from scratch would be extremely costly.

Using cloud-native providers also alleviates some of the responsibility. Google [9], AWS [10] and Azure [11] all have shared responsibility models where they determine who oversees what. This gives a team less to worry about, as with certain packages operating systems, networking and even security patches can all be handled and managed by the cloud provider, dependant on what kind of service you are using.

Finally a cloud-native approach is much more scalable and resilient. There are two ways to scale, vertically and horizontally. Vertically refers to adding more computing power, horizontally refers to adding extra machines [12].

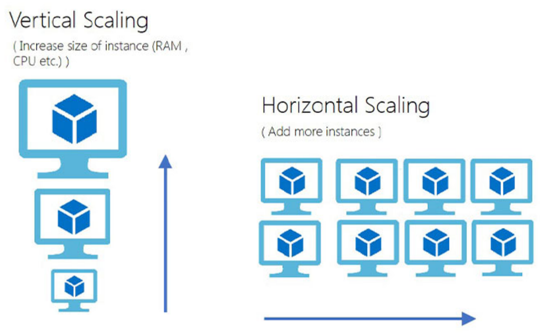


Figure 3: Diagram illustrating the different ways to scale infrastructure. [13]

In an on-prem situation, scaling is expensive in both ways, buying a whole new server is not feasible for ACME, never mind the management of how redundancy takes place. But upgrading the hardware would also not be too cheap either. Cloud providers work at such a large scale that they can offer these features at a fraction of the cost that on-prem can. In addition there are options provided by the cloud providers to have fallbacks for failures and load balancing for quicker response times. These are features that are costly to develop and maintain on ones own.

2.3 Drawbacks of a cloud-native approach

Although there is a lot of positives to cloud-native, there is never a perfect solution. Here is a list of things to consider when adopting a cloud-native approach.

- **External dependency** - Adding another external dependency to a business is another thing that can go wrong. This year the BBC, Boots and others were caught up in an attack that revealed sensitive information about staff [14]. This was done by an attack using an external provider to gain access to the companies using it. Although this is very unlikely, and even with full-control hacks could happen, it's something to consider.

- **Lock in** - Once you've picked a cloud provider to go with, the more infrastructure you build the harder it is to move away. With IaC (Infrastructure as Code) [15] being used in a lot of organisations, it's not just a service switch, it can be the rewriting of 1000s of lines of code. Research is vital here, making sure the organisation you go with has the things you need and is expanding is vital to not reach a situation where you can't build what you want.

- **Lack of control** - You can't control what stays and what gos on the providers platform. They could deprecate systems you were using leaving you with a lot of issues. This has happened in the past with certain version of software, for example node versions being deprecated [16]. The main reason this happens however is because the software is no longer supported by the developers. This could lead to security issues in the future and it is therefore unsafe to use it. In addition to this, features are usually *soft deprecated* which refer to *'an API which should no longer be used to write new code, but it remains safe to continue using it in existing code'* [17].

- **Knowledge -** Cloud development and IAC [15] requires knowledge of how they work and piece together. ACME can put their developers who create the site on courses to learn this or hire a specialist who knows all about it already. This is an additional cost/factor to think about. I don't see this is an issue though, as with the on-prem alternative you also need someone to manage the physical hardware as well as the software running on it.

Despite these potentialities the realities; speed of development, fallbacks, lower cost and an array of services still make cloud-native the best choice.

2.3.1 Sub architectures in a cloud-native approach

As previously mentioned, using a cloud-native approach allows access to multiple services. These services have different architectures that we can take advantage of for individual parts/components of the system. Below breaks down the previous architecture into these separate architectures/services.

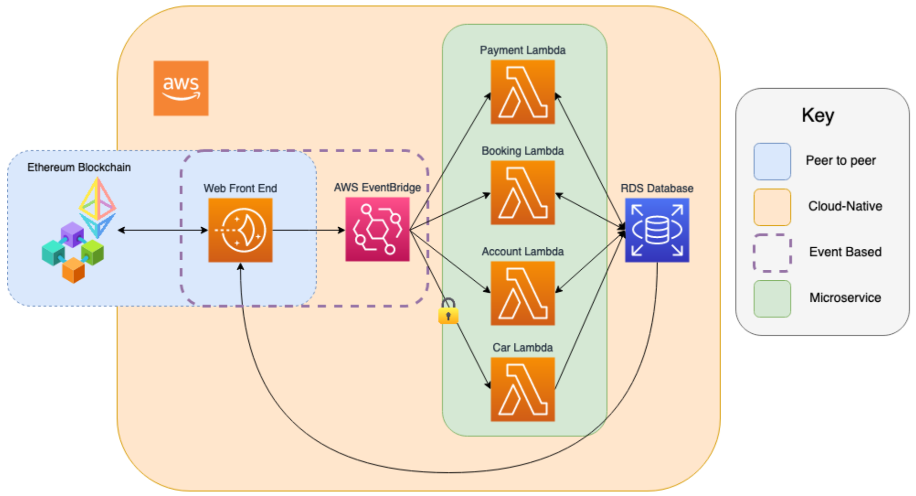


Figure 4: Diagram showing the proposed architecture with the different types of architecture labelled.

- **Cloud-Native** - Cloud-native is the main architecture and *'wraps'* around all the other architectures for the project. The benefits of this architecture have been discussed through this report.

- **Microservices** - Microservices have been mentioned in this report already, however the design illustrates how helpful they can be. The 4 lambdas in the design work independently of each other. If there's an issue with the payment lambda, users can still access the functionality provided by the account lambda. Breaking the structure down this way also makes code more maintainable, as you can have separate repos for each component. A study in 2017 [18] that compared 3 different deployment options, monolithic, microservices and lambda. In this study the microservices architecture still used EC2s/servers but instead had multiple smaller ones doing tasks, ACMEs' new design is more reminiscent of the *'lambda'* implementation, which used different AWS services to achieve its goal. As can be seen in the below, the suggested architecture can not only be cheaper using the microservice/lambda approach, but can actually result in quicker response times for the consumer.

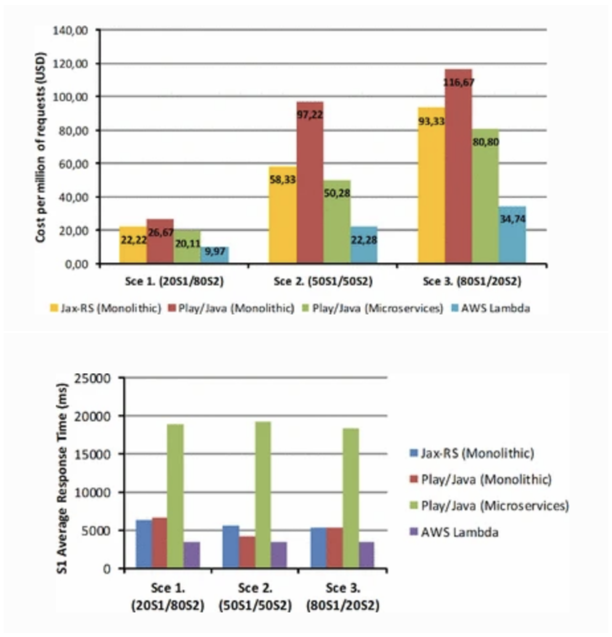


Figure 5: Charts showing results for the 3 solutions [19].

Microservices aren't always the way to go, in fact Amazon themselves swapped away from this architecture for their live audio/video monitoring service [19]. This makes sens as serverless/microservices are not made for continuous running. The microservices in ACMEs design would only run when they received events to do so.

- **Event-Based** - In the above design AWS EventBridge [19] is used to fire events to the individual lambdas. A benefit of this is the responsiveness of the system. The client side has 0 wait time, as the event is fired off to the EventBridge which then routes the message to the correct processor. Website performance is key to customer retention, one of ACMEs' business goals. An article by The Drum states that *'79% of people wouldn't return to a site that had previously performed poorly for them' [20].* Another article quotes:

  > *'Previous research has shown that user frustration increases when page load times exceed eight to 10 seconds, without feedback' [21]*

  This event-driven approach minimises this load time. The EventBridge and its rules also filters out any bad traffic/events people try to send to it resulting in only valid event being processed.

  Some issues with event-based systems is error handling, which I will cover later in the report, and duplicated events. The system would have to have some code built into it to stop people creating multiple orders instead of just the 1. Techniques like debouncing (*'a function ensures that it doesn't get called too frequently.'* [22]) can be used to stop this.

- **Peer-2-Peer** - Finally my system is exposed to the P2P architecture due to the nature of cryptocurrency. Peer to peer can be described as:

  > *'a decentralized platform whereby two individuals interact directly with each other, without intermediation by a third party.'* [23]

  So if a user was to pay with cryptocurrency, there is no Stripe or bank in between ACME and their finances, ACME hs full control over that transaction. Smart contracts [24] can be used to automate and validate payments made, however this requires knowledge of the Solidity [25] programming language.

  In my previous report I spoke about how quickly cryptocurrency adoption is growing. The automation through smart contracts allow and no middlemen make the process easy to implement. Security of these contracts is paramount, so audits of this code should be carried out. Tools such as Chat GPT have been tested to audit smart contracts, with mixed results [26][27].

3 Dependability

In his book *Software Engineering*, Ian Sommerville describes the term dependability to mean *'The dependability of a computer system is a property of the system that reflects its trustworthiness.'* [28]. He the breaks this down into 4 sections which I will now discuss.

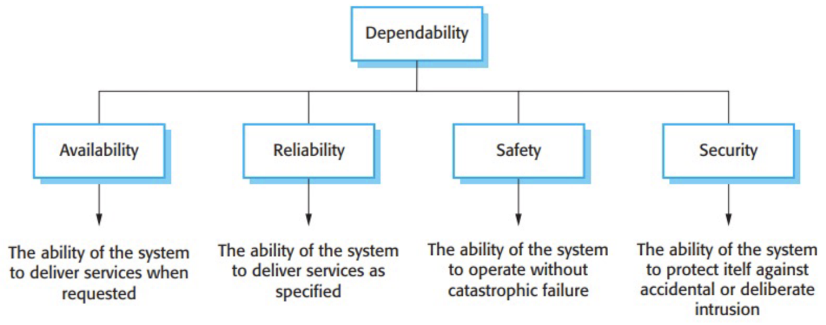

Dependability
Availability
Reliability
Safety
Security
The ability of the system to deliver services when requested
The ability of the system to deliver services as specified
The ability of the system to operate without catastrophic failure
The ability of the system to protect itelf against accidental or deliberate intrusion

Figure 6: Diagram showing the different parts of dependability [28].

3.1 Reliability

Availability and reliability are closely linked. The image below describes the difference.

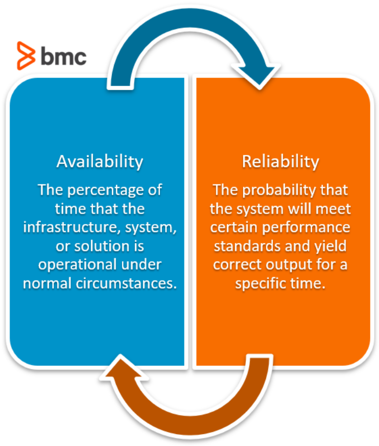

bmc
Availability
The percentage of time that the infrastructure, system, or solution is operational under normal circumstances.
Reliability
The probability that the system will meet certain performance standards and yield correct output for a specific time.

Figure 7: Diagram showing the difference between availability and reliability [29].

3.1.1 Metric for ACME system reliability

Looking at the system designed the main component is the customer web app. This needs to successfully service as many customers as is possible. I have chosen the metric **POFOD = 0.0001** meaning that it is acceptable for the service to be unavailable once every 1000 requests. As this is a cloud-native approach ACME has no control over what AWS will do. They themselves will have systems in place to counteract this however for 99.9% of requests to succeed is still high availability.

To measure this requirement AWS can help us once again. AWS provides metrics for it's components. For the web app we can check the status code metric, if the user receives a 5** this is a server error, otherwise known as downtime. We can then use this data to check the amount of requests that have failed due to internal server errors and check it against the metric.

3.2 Fault tolerance

Other than third party issues there are security problems that could arise. The first thing to tackle is the code itself, the code should be written in a memory safe language. Non-memory safe languages (such as C) allow programmers direct access to memory which can result in attacks such as buffer overflows and use-after-free attacks [30]. Such attacks can end in system failure and sometimes allow a hacker to infiltrate the system. For this reason I would suggest ACME use a language like Python. Not only is it memory safe, but it's got a vibrant community and an older study showed that the total code written and time spent writing was the lowest out of a handful of other languages [31].

Ian Sommerville outlines 8 steps for dependable programming guidelines [28]. I will now go through each one and give examples of how ACMEs' system can handle issues in that area.

1. *Limit the visibility of information in a programme* - This can be linked to the Principle of Least Privilege (PoLP) idea which *'refers to an information security concept in which a user is given the minimum levels of access' [32]*. Users, both internal and external, should only have access to what they need to access to do their jobs.

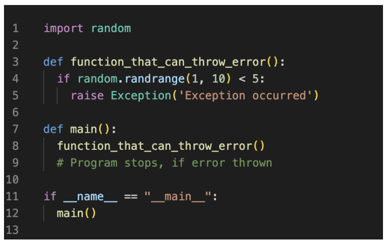
   However this can also be extended in a cloud-native architecture so that the individual components of the system only need permission to access what they need to function. For example the payment lambda does not need access to the account lambda. This not only breaks PoLP, but also the idea of microservices being self-contained operations. Steps and process should be in place and enforced to both staff and code to revoke permissions that are no longer needed.

2. *Check all inputs for validity* - This is vital for two reasons. The first is data integrity, you may have a schema that you want users data to conform to. This can make querying data easier, but also save money as you won't be having to pay for extra data storage. Additional checks such as size and

range checks [28] should be done, and helpful error messages returned to the client so they know what to change.

However the main reason is security. In OWASPs' top 10 vulnerabilities list [33] *'Injection'* is 3rd and covers both XSS (Cross Site Scripting) and SQL injection attacks. These are nearly always exploited via unsanitised inputs. Both client and server-side checks must be made on any data provided by the user.

3. *Provide a handler for all exceptions* - This is achieved by try/catch/finally statements, or the alternative in other languages. If an error occurs but is not caught, then the system could fail. In addition to this catching the error also provides an opportunity for monitoring and finding potential defects in the system. Below shows the code difference between a correctly handled error and not.

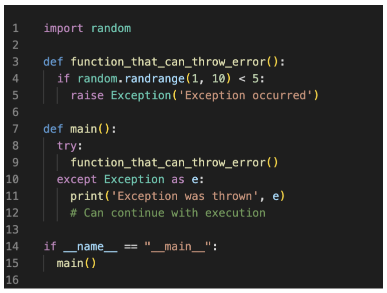


```python
import random

def function_that_can_throw_error():
    if random.randrange(1, 10) < 5:
        raise Exception('Exception occurred')

def main():
    function_that_can_throw_error()
    # Program stops, if error thrown

if __name__ == "__main__":
    main()
```



```python
import random

def function_that_can_throw_error():
    if random.randrange(1, 10) < 5:
        raise Exception('Exception occurred')

def main():
    try:
        function_that_can_throw_error()
    except Exception as e:
        print('Exception was thrown', e)
        # Can continue with execution

if __name__ == "__main__":
    main()
```

Figure 8: Code demonstrating error handling (right) and no error handling (left) using Python.

4. *Minimise the use of error-prone constructs* - A list of these constructs was compiled by Ian Somerville [34], however a lot of these are outdated, and refer to languages like C where memory allocation, array bound checking and pointers all posed issues for the developer when used incorrectly. A principle ACME follows is DRY code [35], which refers to clear concise code and Don't Repeat Yourself. Other issues are poor documentation. If everyone left the team then the new employees need this documentation to fully understand how the system works.

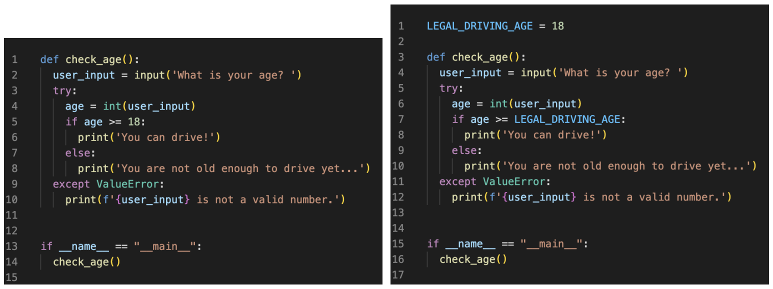
5. *Provide restart capabilities* - This is discussed in the **Rollbacks** section.

6. *Check array bounds* - Memory leaks are no longer an issue in modern languages, but errors can still occur. Some languages like JavaScript allow this access and return a default value instead of erroring. However this still has to be checked, otherwise functions that don't exists could be called, ending in system failure/error.

7. *Include timeouts when calling external components* - This is important as a faulty or even compromised external service could make your own service

unavailable when trying to retrieve data from it. If one user initiated a request to the external resource that took too long another user coming along in that time may not be able to access the service. In addition to this it's frustrating to the user to be sat around waiting when the data may never be returned. Most libraries allow developers to add timeouts to any network request made and this is implemented into the ACME system.

8. *Name all constants that represent real-world values* - This is a good practice and is sometime referred to as *'magic numbers'* [36]. This can be troublesome when these hard-coded values are used in multiple places. ACME uses config files for these values that can then be imported throughout the project. This makes the code easier to change and more readable.



```python
def check_age():
    user_input = input('What is your age? ')
    try:
        age = int(user_input)
        if age >= 18:
            print('You can drive!')
        else:
            print('You are not old enough to drive yet...')
    except ValueError:
        print(f'{user_input} is not a valid number.')


if __name__ == "__main__":
    check_age()
```

```python
LEGAL_DRIVING_AGE = 18

def check_age():
    user_input = input('What is your age? ')
    try:
        age = int(user_input)
        if age >= LEGAL_DRIVING_AGE:
            print('You can drive!')
        else:
            print('You are not old enough to drive yet...')
    except ValueError:
        print(f'{user_input} is not a valid number.')


if __name__ == "__main__":
    check_age()
```

Figure 9: Code demonstrating magic numbers (left) and without (right) using Python.

3.2.1 Rollbacks

AWS uses metrics to track events, these can be both default and custom metrics defined in code. These metrics can then be hooked up to alarms which when state changes can trigger events that are handled by the EventBridge currently in the architecture [37]. These events can then be used to trigger additional actions such as code rollbacks or additional checks to determine/fix issues.

Rollbacks could be made possible by using Jenkins [38], which is a CI/CD provider. This kind of action should only be taken on components that render the service unusable. This automation of rolling back could have other consequences to other systems and therefore should be a last resort and very well tested. For example if a microservice failed, it might not be worth having this automatic process. However if the web front end encountered system failure it'd be worth auto restarting, especially when there is no staff currently working to investigate the problems themselves.

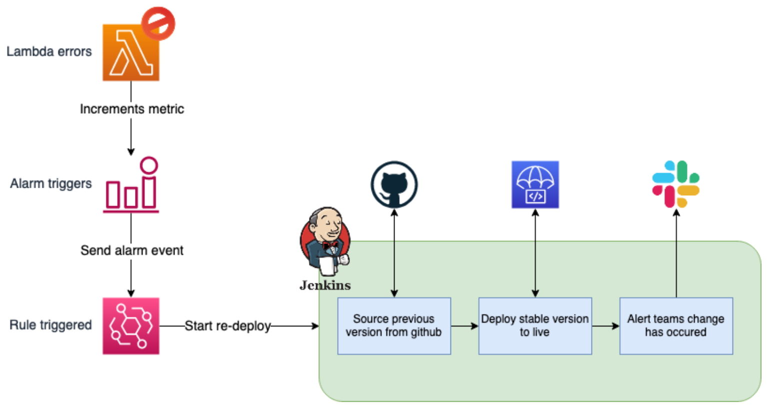


Figure 10: Diagram showing how the CI/CD pipeline of a rollback system would operate.

3.3 Difference between safety and security threats

Safety and security are not the same things when it comes to software, a definition for safety is a:

> *'should never damage people or environment even though the system fails'* [39]

Whereas security can described as:

> *'a system attribute that reflects the ability of the system to protect itself from external attacks, which may be accidental or deliberate.'* [39]

Despite this there are situations in which these two can cross over fully. A german hospital was hacked and a woman ended up dying due the hack forcing her to be transported to another hospital. This was deemed *'the first known case of a life being lost as a result of a hack'* [40]. In the rest of this section I will discuss how ACMEs' system protects/avoids the two above issues.

3.4 Safety

Safety of a system can often times be linked to harming other peoples health. The system ACME has created can't directly do that, however could have indirectly. One issue is financial, a bug in the system could severely harm someone financially. There is also a chance cars provided are faulty or the person driving them provided fake ID to gain access to the vehicle, then causing others harm.

Lets starts by discussing financial issues, I previously talked about debouncing being a way to stop users accidentally performing an action more than once. Other ways include making sure a user has to confirm there purchase after the initial action can stop accidental purchases. However mistakes can still happened, therefore ACME offers email and phone channels where customers can communicate with staff to get refunded.

To verify drivers are who they say they are ACME has multiple checks before keys are given to a customer. The UK government offers a service to check the validity of individuals driving licenses [41], which will be used when the customer creates an account. Adding further security to this, drivers licenses must be checked when the customer picks up the car. If something isn't correct, the customer is refunded and the car is not rented to them.

Doing the above will help with safety concerns that could arise from ACME's system. There is always a chance something will happen that is out of ACMEs' control, however if the correct steps are taken these occurrences can be minimised.

3.5 Security

I now will discuss security in two categories, physical (real-world) and software.

3.5.1 Physical

Some threats of physical security are reduced due to the cloud-native approach chosen. This is due to no on-prem servers being used, therefore user data is not accessible through ACME. However there are still some threats worth discussing.

- *Tailgating* - This is a method where an *'unauthorized person gains physical access to an off-limits location'* [42]. This can lead this individual gaining access to machines or information they shouldn't have access to. Staff training is required here as well as tight security policies around passes. Gates should only allow one person through at a time and staff should report any suspicious behaviour.

- *Physical access to machines* - This can be garnered by the above method or other methods of social engineering/distraction. This shouldn't be too hard to prevent, if devices like PCs are locked when the user is not there. In addition, ACME will have a password policy that makes sure that passwords are changed every 3 months as well meet certain criteria, such as length and special characters contained. I don't recommend a password manager despite their ease of use as this represents a single point of failure [43]. Tools like haveibeenpwned [44] can also be used to determine whether someones email has been in a recent security breach.

3.5.2 Software

I have spoke throughout this report about software security risks that can occur if developers are not careful. In this section I want to discuss more about preventative measures that can be taken to avoid software issues from happening.

- **Penetration testing** - This *'is a cyberattack simulation launched on your computer system'* [45] most often carried out by a third party. There are 3 different approaches to this, black, grey and white box [46].

Type	**Access Granted**
Black Box	None
Grey Box	Some
White Box	Full Access

Table 1: Table showing pen testers access using different methodologies.

  I would recommend a mixture of black and white box pen testing. Black box simulates an attacked on the outside trying to gain access to the system. Whilst a white box test gives the pen tester the chance to view the inner workings of the code. This could flag any potential vulnerabilities that made it through the QA/testing phase. This should be done once a year, as these services can cost *'anywhere between £600 to over £3000 per day'* [47].

- **Principle of least privilege (PoLP)** - As previously **menitioned** ACME will incorporate processes for both onboarding and off-boarding new staff members. No staff member should have any privileges they do not need to do their job. Although this can seem to be untrusting, however with 34% of businesses suffering attacks from insiders [48] and 66% of companies being more worried about an internal attack than a external attack [48] highlights the potential issue.

- **QA validation/testing** - The value of testing is discussed more in the **Testing** section.

- **Code reviews** - Code reviews act as a *'second opinion'* [49] on newly implemented code. Another developer should always read any code that plans to be released to a live environment to help stop bugs reaching consumers.

- **Strict dependency selection** - ACME should use only well-known and tested external dependencies. In addition to this these should be checked into code so other developers can review them before live release. There have been multiple instances [50] where packages have been *'typo-squatted'* [51] and made their way to consumers.

- **Firewalls/blacklists** - AWS offers services such as WAF [52] to help protect customers against some forms of attacks. An extra layer can be added to this. Software can be written to determine malicious or concerning usage of the web app, such as excessive requests (DDos) attack3[53]. These can then be put onto a blacklist and their requests can be dropped from then on.

4 Is the project V & V approved?

V & V refers to *'Validation'* and *'Verification'*. In this section I will discuss why I think this project is v & v approved as well as discuss more about the deployment and testing of the new system.

> *'The aim of validation is to ensure that the software meets the customer's expectations'* [54]

> *'The aim of verification is to check that the software meets its stated functional and non-functional requirements'* [54]

4.1 Customer Feedback

To validate the product has met customer expectations, I think it's important to first note that the old system of phoning/emailing is still in place for customers who don't want to transition to the new system. Systems on the web front-end could be put into place in the future to help gain feedback of how the customer experience was and how it could be improved.

In the previous report I spoke about how using the Agile software development lifecycle would also help keep customers/stakeholders involved in the development process. This way of working also allows for quick changes, so if there ever is something inefficient or sub-standard with the service the development team can quickly get to work at implementing the new changes.

4.2 Testing

When software is not tested issues such as poor quality, performance and user experience, security vulnerabilities and increased maintenance cost [55] can all occur.

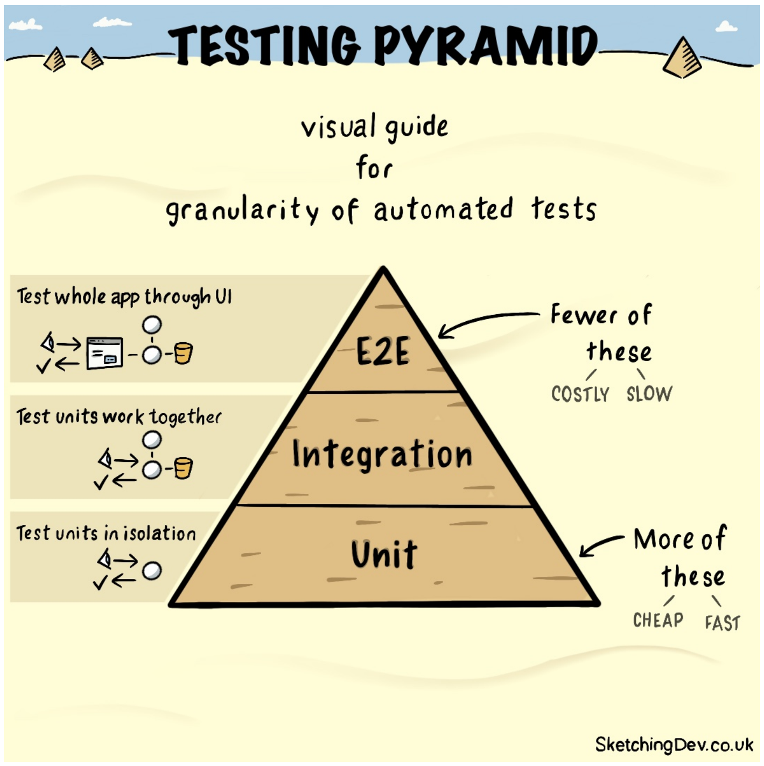


Figure 11: Image showing the testing pyramid, representing types of test and their quantity. [56]

The testing pyramid above shows the types of testing ACME performs as well as demonstrates the quantity of each type of test.

- **Unit Tests** - These can be described as *'consist[ing] in testing individual methods and functions of the classes, components, or modules'* [57]. A simple example of a unit test would be to make sure a function that adds two number together, provides the expected result when given two numbers. Mocking, *'replacement object for [a] dependency'* [58] are often used in these types of test to remain cheap by stopping expensive calls to databases or web requests by providing a pre-programmed response that the test can use instead. Due to their lower cost unit tests can be ran before every deployment.
- **Integration Tests** - Unlike unit tests, integration tests want to test the actual functionality of the system, thus making them more costly. In other words *'Integration tests verify that different modules or services used by your application work well together'* [57]. These are run on a schedule rather than at deployment to help lower the overall expenditure of testing.

- **End-to-end (E2E) Tests** - End-to-end tests, also known as UI tests, *'replicates a user behavior with the software'* [57]. This is done on software that is either live, or functions the exact same way the live system does. This is the most expensive type of testing as it often requires a large overhead to run. For example a library like Puppeteer [59] uses the chromium browser to make network requests to a specified test url, then replicates user actions and runs assertions based on those actions. This is a lot more CPU intensive work than the previous tests, and like integration tests should be run on a schedule.

A breakdown of ACMEs testing can be seen in **AppendixC**.

4.3 Deployment

Deployment can be described as the *'process of putting software and software solutions into use or action'* [60]. This stage can be used to run tests, build and of course deploy the final application. Below is the jenkins pipeline ACME uses to deploy their app.

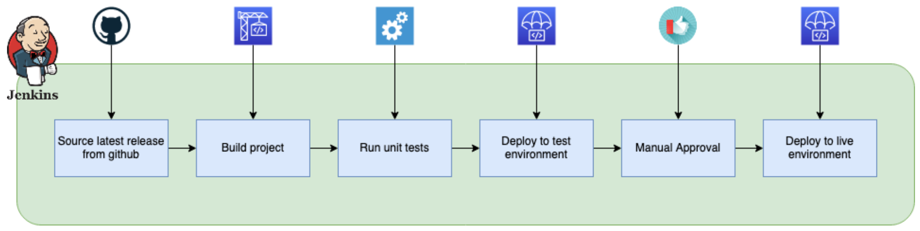

Jenkins
Source latest release from github
Build project
Run unit tests
Deploy to test environment
Manual Approval
Deploy to live environment

Figure 12: Diagram showing the stages of release for project.

If at any point one of the stages fails, the pipeline fails and the following stages are not executed. This is helpful as if the unit tests fail this would indicate an issue and therefore we wouldn't want the broken system to be deployed to live. I have also set it up in a way where there are two environments. First the new code is deployed to test, where it can be double checked and made sure that it is ready by testers. Then, with manual approval, it can be deployed to the live environment. The idea here is having this separation of environments will help catch bugs on the real hardware before going to live.

5 Conclusion

To conclude, in this report I have discussed the architecture that ACME uses in its newly created system. A cloud-native approach is the best approach when considering ACME prior paper-based system, due to setup cost and speed. This system is V & V approved due to its rigorous, testing, deployment and requirements gathering to ensure what's being built is correct and of high quality.

In addition to this the system also has protections for safety and security concerns, some of which can be provided by AWS, but also internal systems such as IP/user blacklists ans support channels on the web app. The language and packages used are modern and well tested preventing certain attacks like buffer overflows out the box.

In addition to this, the adoption of an agile approach allows the team to make changes in the future to make their service even better for the customer.

6 References

Cloud Migration - (Oracle) `https://www.infosys.com/Oracle/white-papers/Documents/cloud-migration-assessment-framework.pdf`
[1] Sommerville, I. (2020). Engineering Software Products: An Introduction to Modern Software Engineering. Global edition. USA:Pearson.

[2] StackOverflow. (2023) *Stack Overflow Developer Survey 2023* [online]. Available at `https://survey.stackoverflow.co/2023/#cloud-platforms` (accessed 11th August).

[3] Amazon Web Services. (2023) *What is Cloud Native? - Everything you need to know - AWS* [online]. Available at `https://aws.amazon.com/what-is/cloud-native/` (accessed 11th August).

[4] IBM. (2023) *What is Virtualization? — IBM* [online]. Available at `https://www.ibm.com/topics/virtualization` (accessed 11th August).

[5] Firesmith, D. (2017) *Virtualization via Virtual Machines* [online]. Available at `https://insights.sei.cmu.edu/blog/virtualization-via-virtual-machines/` (accessed 11th August).

[6] Amazon Web Services. (2023) *Free Cloud Computing Services - AWS Free Tier* [online]. Available at `https://aws.amazon.com/free/` (accessed 11th August).

[7] IBM. (2023) *What are microservices? — IBM* [online]. Available at `https://www.ibm.com/topics/microservices` (accessed 11th August).

[8] Amazon Web Services. (2023) *Global Infrastructure Regions & AZs* [online]. Available at `https://aws.amazon.com/about-aws/global-infrastructure/regions_az/` (accessed 11th August).

[9] Alphabet Inc. (2023) *Shared responsibilities and shared fate on Google Cloud — Architecture Framework* [online]. Available at `https://cloud.google.com/architecture/framework/security/shared-responsibility-shared-fate` (accessed 11th August).

[10] Amazon Web Services. (2023) *Shared Responsibility Model - Amazon Web Services (AWS)* [online]. Available at `https://aws.amazon.com/compliance/shared-responsibility-model/` (accessed 11th August).

[11] Microsoft. (2023) *Shared responsibility in the cloud - Microsoft Azure — Microsoft Learn* [online]. Available at `https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility` (accessed 11th August).

[12] Ozkaya, M. (2021) *Scalability — Vertical or Horizontal Scaling when Designing Architectures* [online]. Available at `https://medium.com/design-microservices-architecture-wit`
`scalability-vertical-scaling-horizontal-scaling-adb52ff679f` (accessed 11th August).

[13] WebAiry. (2019) *Horizontal and Vertical Scaling* [online]. Available at `https://www.webairy.com/horizontal-and-vertical-scaling/` (accessed 11th August).

[14] Tidy, J. (2023) *MOVEit hack: BBC, BA and Boots among cyber attack victims* [online]. Available at `https://www.bbc.co.uk/news/technology-65814104` (accessed 11th August).

[15] IBM. (2023) *What is Infrastructure as Code (IaC)?* [online]. Available at `https://www.ibm.com/topics/infrastructure-as-code` (accessed 11th August).

[16] Amazon Web Services. (2023) *Announcing the end of support for Node.js 12.x in the AWS SDK for JavaScript (v3)* [online]. Available at `https://aws.amazon.com/blogs/developer/announcing-the-end-of-support-for-node-js-12-x-in-the-aws-sdk-f` (accessed 11th August).

[17] Peterson, B. (2023) *PEP 387 - Backwards Compatibility Policy* [online]. Available at `https://peps.python.org/pep-0387/` (accessed 11th August).

[18] Villamizar, M., Oscar Garcés, et al. (2017) *Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures* [online]. Available at `https://link.springer.com/article/10.1007/s11761-017-0208-y#Sec15` (accessed 14th August).

[19] Kolny, M. (2023) *Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%* [online]. Available at `https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-` (accessed 14th August).

[20] Solanki, R. (2019) *Why a slow website is killing your conversions* [online]. Available at `https://www.thedrum.com/opinion/2019/08/28/why-slow-website-killing-your-conversi` (accessed 14th August).

[21] Website Optimization. (2008) *The Psychology of Web Performance - how slow response times affect user psychology* [online]. Available at `https://www.websiteoptimization.com/speed/tweak/psychology-web-performance/` (accessed 14th August).

[22] Charles, J. (2018) *What is Debouncing?* [online]. Available at `https://medium.com/@jamischarles/what-is-debouncing-2505c0648ff1` (accessed 14th August).

[23] Baldwin, A. (2021) *Peer-to-Peer in Blockchain: how it works* [online]. Available at `https://www.cryptopolitan.com/peer-to-peer-in-blockchain-how-it-works/` (accessed 14th August).

[24] IBM. (2023) *What are smart contracts on blockchain?* [online]. Available at `https://www.ibm.com/topics/smart-contracts` (accessed 14th August).

[25] Solidity Team. (2023) *Home — Solidity Programming Language* [online]. Available at `https://soliditylang.org/` (accessed 14th August).

[26] Alici, I, U. et al. (2023) *OpenAI ChatGPT for Smart Contract Security Testing: Discussion and Future Directions* [online]. Available at https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4412215 (accessed 14th August).

[27] QuillAudits Team. (2023) *Beyond the Hype: ChatGPT and Smart Contract Auditing QuillAudits Team* [online]. Available at https://blog.quillaudits.com/2023/04/03/beyond-the-hype-chatgpt-and-smart-contract-auditing/ (accessed 14th August).

[28] Sommerville, I. (2016). Software Engineering. Tenth edition. USA:Pearson.

[29] Raza, M. (2020) *Reliability vs Availability: What's The Difference?* [online]. Available at https://www.bmc.com/blogs/reliability-vs-availability/ (accessed 14th August).

[30] Caballar, D, R. (2023) *The Move to Memory-Safe Programming* [online]. Available at https://spectrum.ieee.org/memory-safe-programming-languages (accessed 14th August).

[31] Prechelt, L. (2000) *An Empirical Comparison of Seven Programming Languages* [online]. Available at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.1831&rep=rep1&type=pdf (accessed 14th August).

[32] CyberArk. (2023) *What is Least Privilege? Principle of Least Privilege Definition* [online]. Available at https://www.cyberark.com/what-is/least-privilege/ (accessed 15th August).

[33] OWASP. (2021) *OWASP Top Ten* [online]. Available at https://owasp.org/www-project-top-ten/ (accessed 15th August).

[34] Sommerville, I. (2019) *Error prone constructs* [online]. Available at https://software-engineering-book.com/web/error-prone-constructs/ (accessed 15th August).

[35] Munoz, D. (2021) *What is DRY Code* [online]. Available at https://codinglead.co/javascript/what-is-DRY-code (accessed 15th August).

[36] Fulber-Garcia, V. (2023) *Antipatterns: Magic Numbers* [online]. Available at https://www.baeldung.com/cs/antipatterns-magic-numbers (accessed 15th August).

[37] Amazon Web Services. (2023) *Alarm events and EventBridge* [online]. Available at https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch-and-eventbridge.html (accessed 15th August).

[38] Jenkins. (2023) *Jenkins* [online]. Available at https://www.jenkins.io/ (accessed 15th August).

[39] Roy, D. (2023) *Safety and Security Engineering* [online]. Available at https://vle.aston.ac.uk/bbcswebdav/pid-3316291-dt-content-rid-24421304_

1/xid-24421304_1 (accessed 15th August).

[40] Tidy, J. (2020) *Police launch homicide inquiry after German hospital hack* [online]. Available at https://www.bbc.co.uk/news/technology-54204356 (accessed 15th August).

[41] UK Government. (2023) *Check someone's driving licence information* [online]. Available at https://www.gov.uk/check-driving-information (accessed 16th August).

[42] McAfee LLC. (2023) *What Are Tailgating Attacks and How to Protect Yourself From Them* [online]. Available at https://www.mcafee.com/blogs/internet-security/what-are-tailgating-attacks/ (accessed 16th August).

[43] Winder, D. (2022) *LastPass Hacked: Password Manager With 25 Million Users Confirms Breach* [online]. Available at https://www.forbes.com/sites/daveywinder/2022/08/25/lastpass-hacked-password-manager-with-25-million-users-confirm (accessed 16th August).

[44] Hunt, T. (2023) *Have I Been Pwned: Check if your email has been compromised in a data breach* [online]. Available at https://haveibeenpwned.com/ (accessed 16th August).

[45] Cisco Systems Inc. (2023) *What Is Penetration Testing?* [online]. Available at https://www.cisco.com/c/en/us/products/security/what-is-pen-testing.html#~types-of-pen-testing (accessed 22nd August).

[46] Poston, H. (2020) *What are black box, grey box, and white box penetration testing?* [online]. Available at https://resources.infosecinstitute.com/topics/penetration-testing/what-are-black-box-grey-box-and-white-box-penetration-testing/ (accessed 22nd August).

[47] Evalian. (2023) *Penetration testing costs: A comprehensive guide* [online]. Available at https://evalian.co.uk/a-comprehensive-guide-to-understanding-penetration-testing- (accessed 22nd August).

[48] Chekalov, M. (2023) *22 Insider Threat Statistics to Look Out For in 2023* [online]. Available at https://techjury.net/blog/insider-threat-statistics/ (accessed 22nd August).

[49] GitLab B.V. (2023) *What is a code review?* [online]. Available at https://about.gitlab.com/topics/version-control/what-is-code-review/ (accessed 23rd August).

[50] Lakshmanan, R. (2023) *11 Malicious PyPI Python Libraries Caught Stealing Discord Tokens and Installing Shells* [online]. Available at https://thehackernews.com/2021/11/11-malicious-pypi-python-libraries.html (accessed 23rd August).

[51] AO Kaspersky Lab. (2023) *What is Typosquatting?* [online]. Available at https://www.kaspersky.com/resource-center/definitions/what-is-typosquatting

(accessed 23rd August).

[52] Amazon Web Services. (2023) *Web Application Firewall, Web API Protection - AWS WAF - AWS* [online]. Available at `https://aws.amazon.com/waf/` (accessed 23rd August).

[53] Cloudflare Inc. (2023) *What is a distributed denial-of-service (DDoS) attack? — Cloudflare* [online]. Available at `https://www.cloudflare.com/en-gb/learning/ddos/what-is-a-ddos-attack/` (accessed 23rd August).

[54] Roy, D. (2023) *Deployment and Software Evolution* [online]. Available at `https://vle.aston.ac.uk/bbcswebdav/pid-3327591-dt-content-rid-24486116_1/xid-24486116_1` (accessed 21st August).

[55] Parwal, R. (2023) *The Risks of Performing No Testing or Minimal Testing* [online]. Available at `https://muuktest.com/blog/not-testing-software/` (accessed 21st August).

[56] Sketching Dev. (2023) *Testing Pyramid* [online]. Available at `https://sketchingdev.co.uk/sketchnotes/testing-pyramid.html` (accessed 21st August).

[57] Pittet, S. (2023) *The different types of software testing* [online]. Available at `https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing` (accessed 21st August).

[58] Microsoft. (2023) *Mocking in Unit Tests* [online]. Available at `https://microsoft.github.io/code-with-engineering-playbook/automated-testing/unit-testing/mocking/` (accessed 21st August).

[59] Google, Inc. (2023) *Puppeteer* [online]. Available at `https://pptr.dev/` (accessed 21st August).

[60] Bierds, B. Gibson, J, et al. (2004). The Software Deployment Mystery - Solved: A Customer Guide. USA:IBM.

[61] Consensys. (2023) *The crypto wallet for Defi, Web3 Dapps and NFTs — MetaMask* [online]. Available at `https://metamask.io/` (accessed 14th August).

[62] Amazon Web Services. (2023) *What Is Amazon EventBridge?* [online]. Available at `https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html` (accessed 14th August).

[63] Amazon Web Services. (2023) *What is AWS Lambda?* [online]. Available at `https://docs.aws.amazon.com/lambda/latest/dg/welcome.html` (accessed 14th August).

[64] Stripe Inc. (2023) *Stripe — Payment Processing Platform for the Internet* [online]. Available at `https://stripe.com/gb` (accessed 14th August).

[65] Frankenfield, J. (2022) *Gas (Ethereum): How Gas Fees Work on the Ethereum Blockchain* [online]. Available at `https://www.investopedia.com/terms/g/`

`gas-ethereum.asp` (accessed 21st August).

7 Appendix

7.1 Appendix A - Previous design work

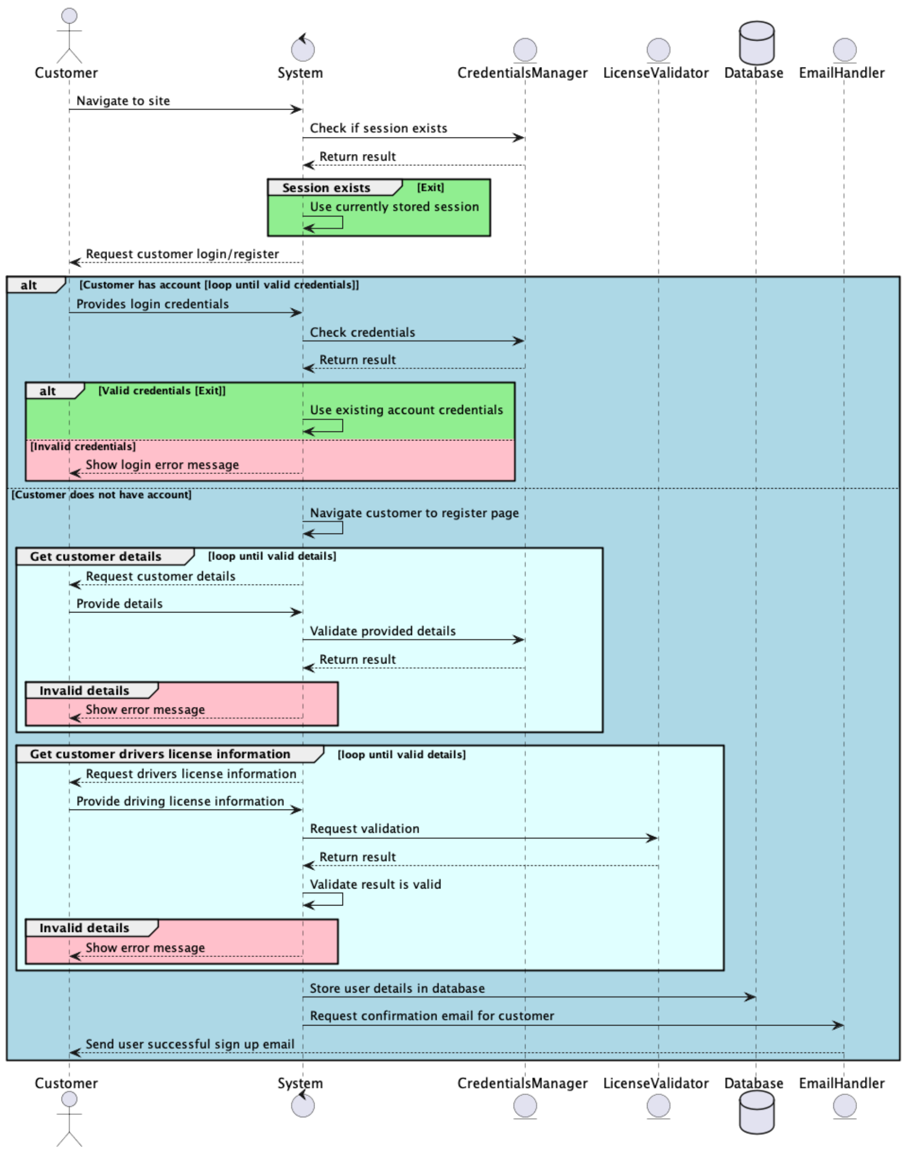

Customer
System
CredentialsManager
LicenseValidator
Database
EmailHandler
Navigate to site
Check if session exists
Return result
Session exists [Exit]
Use currently stored session
Request customer login/register
alt [Customer has account [loop until valid credentials]]
Provides login credentials
Check credentials
Return result
alt [Valid credentials [Exit]]
Use existing account credentials
[Invalid credentials]
Show login error message
[Customer does not have account]
Navigate customer to register page
Get customer details [loop until valid details]
Request customer details
Provide details
Validate provided details
Return result
Invalid details
Show error message
Get customer drivers license information [loop until valid details]
Request drivers license information
Provide driving license information
Request validation
Return result
Validate result is valid
Invalid details
Show error message
Store user details in database
Request confirmation email for customer
Send user successful sign up email

Figure 13: Sequence diagram for adding a new user, this includes sign in/up.

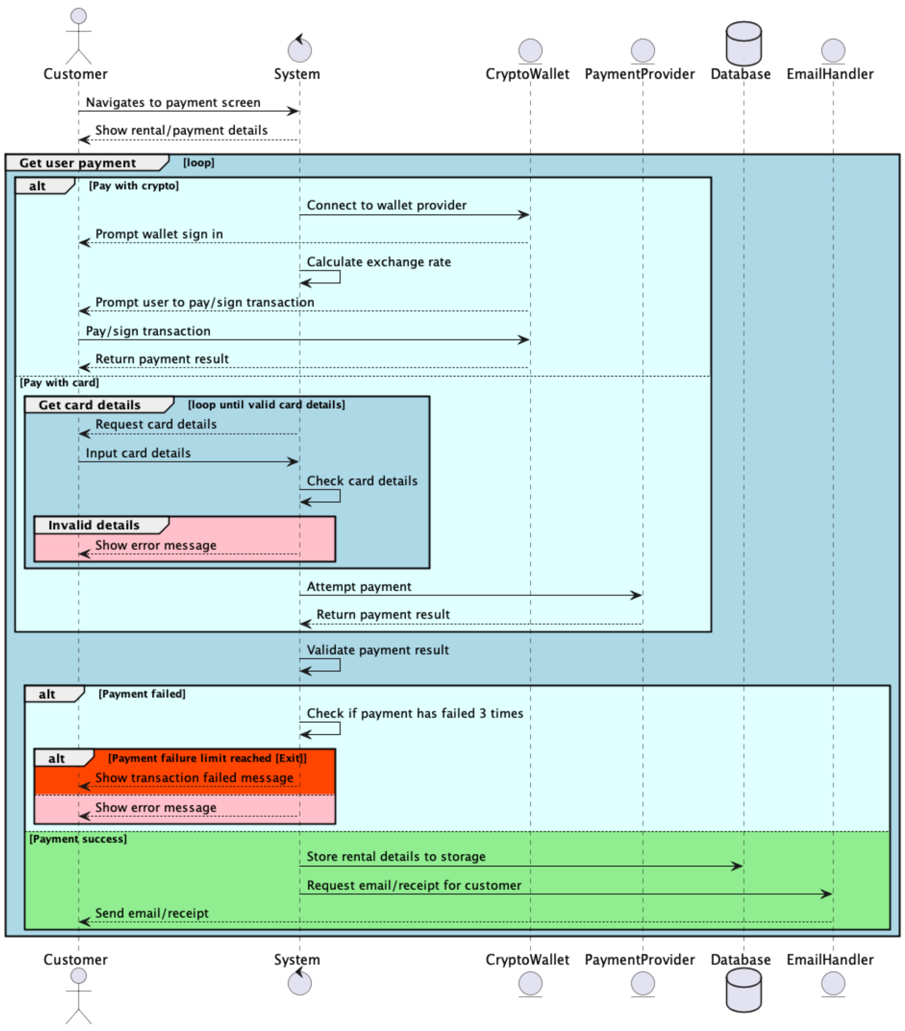


Figure 14: Sequence diagram for taking a payment.

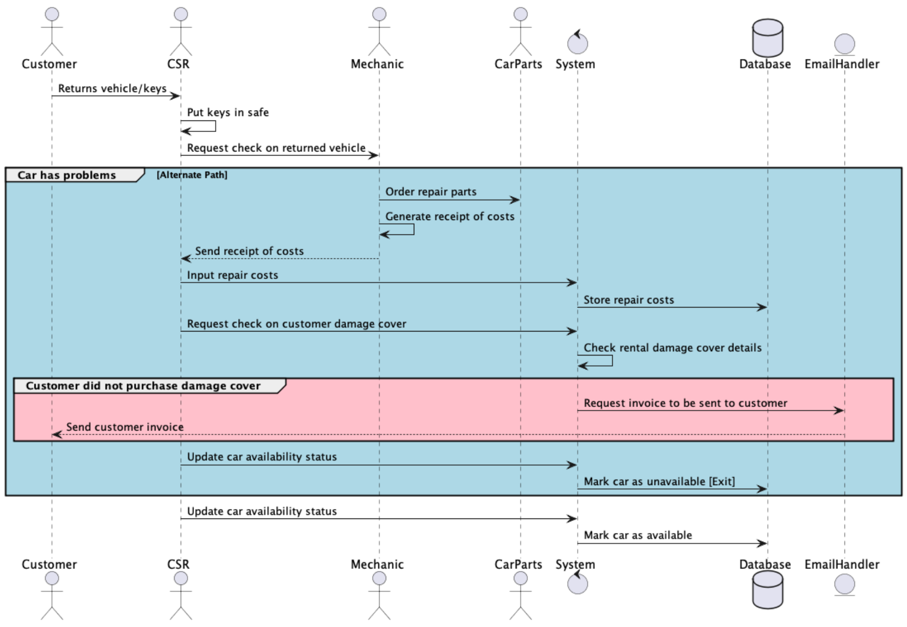

Customer
CSR
Mechanic
CarParts
System
Database
EmailHandler
Returns vehicle/keys
Put keys in safe
Request check on returned vehicle
Car has problems
[Alternate Path]
Order repair parts
Generate receipt of costs
Send receipt of costs
Input repair costs
Store repair costs
Request check on customer damage cover
Check rental damage cover details
Customer did not purchase damage cover
Request invoice to be sent to customer
Send customer invoice
Update car availability status
Mark car as unavailable [Exit]
Update car availability status
Mark car as available

Figure 15: Sequence diagram for handling the return of a vehicle.

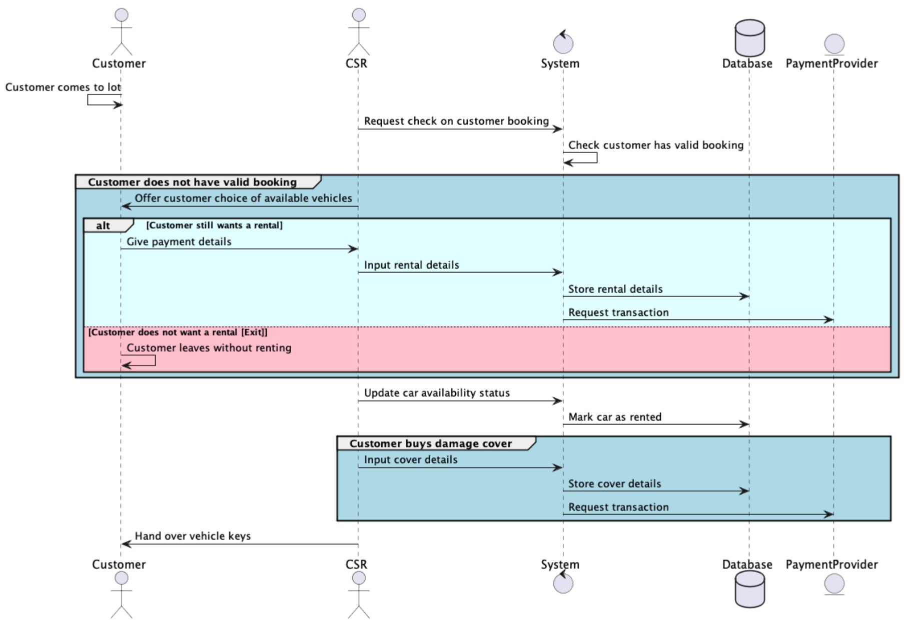

Customer
CSR
System
Database
PaymentProvider
Customer comes to lot
Request check on customer booking
Check customer has valid booking
Customer does not have valid booking
Offer customer choice of available vehicles
alt
[Customer still wants a rental]
Give payment details
Input rental details
Store rental details
Request transaction
[Customer does not want a rental [Exit]]
Customer leaves without renting
Update car availability status
Mark car as rented
Customer buys damage cover
Input cover details
Store cover details
Request transaction
Hand over vehicle keys

Figure 16: Sequence diagram for starting a new hire.

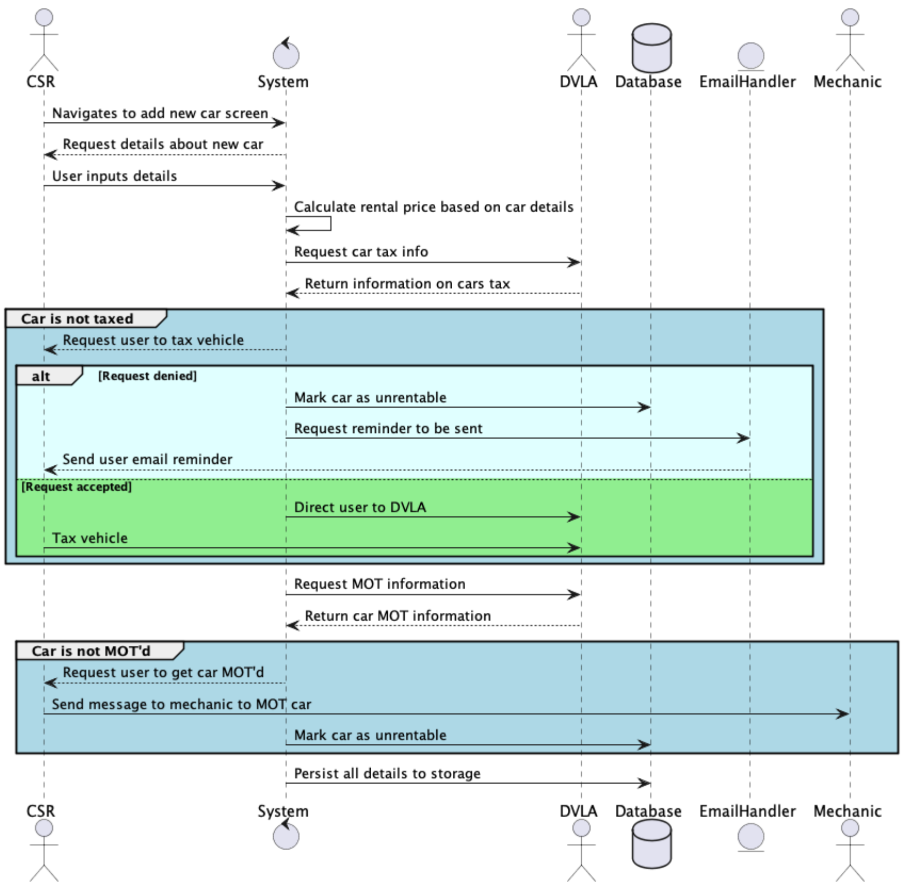

CSR
System
DVLA
Database
EmailHandler
Mechanic
Navigates to add new car screen
Request details about new car
User inputs details
Calculate rental price based on car details
Request car tax info
Return information on cars tax
Car is not taxed
Request user to tax vehicle
alt
[Request denied]
Mark car as unrentable
Request reminder to be sent
Send user email reminder
[Request accepted]
Direct user to DVLA
Tax vehicle
Request MOT information
Return car MOT information
Car is not MOT'd
Request user to get car MOT'd
Send message to mechanic to MOT car
Mark car as unrentable
Persist all details to storage

Figure 17: Sequence diagram for adding a new car to the system.

7.2 Appendix B - Architectural design of system using AWS

This solution will be built primarily using a cloud-native approach, however as this system is somewhat large there is room for other architectural patterns to be used in sub systems of the overall build. Below is a high level look at how the system could be create using AWS:

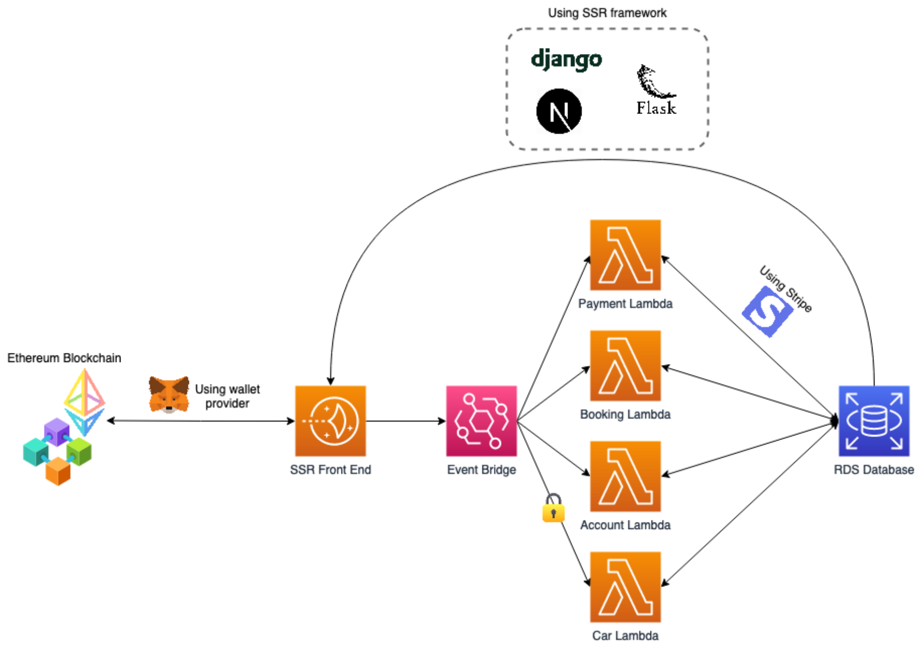


Figure 18: Diagram showing the proposed architecture for the whole system.

The main component is the Server-side Rendered front end. This can then communicate with the blockchain for crypto transactions using a wallet provider, e.g. Metamask [61]. The main functionality is that the front end is then able to send events through EventBridge [62] to individual lambdas [63] for different functionality. These lambdas can then store data into the database. The payment lambda uses Stripe [64] to process payments, and the *'car lambda'* is for internal use only, hence the lock on its connection. These topics will be discussed more in the **Dependability** section.

7.3 Appendix C - ACMEs testing strategy

Test Name	**Test Type**	**Cadence**
UI tests	E2E	Twice a week
Booking tests	Integration	Daily
Payments tests	Integration	Daily
Account tests	Integration	Daily
Car tests	Integration	Daily
P2P/Blockchain tests	E2E	Weekly
Unit tests	Unit	On deployment

Table 2: Table showing ACMEs tests and testing pattern

The above table shows ACME's testing patterns. Most is quite self-explanatory, the only strange one is that P2P/Blockchain and UI tests are not on the same cadence. This was a decision made due to cost. Blockchains require a gas fee [65] when transacting on them. If this is done on a live environment this is going to cost money to pay that fee. Therefore I have lowered this testing suite down to once a week instead of twice.