Beforehand

- Setup all tabs
    - WebGL
    - Dry Rock
- Relax

Presentation

Hi, slides for this talk are available at this URL or via this QR code. For anyone who wants to follow along with the talk.

Next slide

Introduction

So a bit about me. My name is Rory. I'm from Ireland, I've been in Seattle just over 2 years now. I did an undergraduate degree in mathematics. That's where I first dabbled in programming. After college I taught myself how to code more seriously. I've now been working as a software engineer for the last 4 years, mostly in full stack web development. The main languages I've worked with are TypeScript, C#, Python, SQL, and Rust. My other hobbies include rock climbing and skiing.

Next slide

Overview

I just want to give a quick overview of what I will be talking about in this presentation. I will start off by presenting on several projects I have worked on over the years.

- Namely, Atlas, a large project I worked on at my previous job.
- A ray tracing engine that I built in Rust.
- Some open source work that I have done.
- And finally why I want to work at the Allen Institute and this job.

Next slide

Abbey Capital

My previous job was at a company called Abbey Capital. They are a financial fund manager, in other words they manage large sums of other people's money, deciding on the best way to invest that money. They specialise in 'alternative investments'. In their case mostly futures contracts, and other derivatives based on the value of physical commodities.

While I was there I worked as a software engineer building and maintaining internal tooling for the company. I started off mostly handling tickets on existing applications but worked my way up to larger projects.

Next slide

Atlas

The first project I want to talk about today is called Atlas. It was the main project I worked on at Abbey Capital. It was a full re-build of their internal fund accounting application. It was a web application built with a TypeScript React frontend, C# ASP.NET on the backend, and a MySQL database. All running on AWS.

I was part of a small team working on this project, there were just two of us initially, though more people were brought on later, so I was deeply involved with all aspects of the project. We were building this for an internal department at the company so we had direct access to our users and I learnt a lot from sitting with them and watching them use the software.

Next slide

Atlas - functionality

The primary function of Atlas was to calculate daily profit and loss for the funds Abbey managed. It also compared this data to various third parties to ensure their systems were up to scratch. If it found any discrepancies it would flag this for a user to follow up with.

It was designed to improve upon the existing system and automate many manual checks that were taking place.

It did this for 6 different funds, each with their own rules and quirks, and about 7bn dollars in assets.

Next slide

Atlas - architecture

Just to give a little more perspective on this I have a quick diagram of the architecture. Unfortunately I cannot actually show you the application as it is all proprietary.

These are the three main layers of the application, we have the database which stores all of the application data, a backend server, and frontend UI that users interact with.

The backend server did most of the work for this application; serving the frontend UI, running calculations on data, pushing real-time updates to users, and communicating with other services.

Show next step

On that note, I should point out that Atlas was part of a larger set of distributed services. It interacted with other services that we built like the data ingestion tool you can see here and reporting service over here. It generally interacted with these services via message queues and other off the shelf AWS components.

Next slide

Atlas - project structure

The project was split into five main parts:

- **Specification:** During specification we spent several months working with users, to see what they were expecting from the project. Since we were re-building an existing application we used that as a starting point, adding in additional features and making improvements as necessary. We used a tool (Balsamiq) to create visual mock-ups and aid in the design process. This is a technique that I thought was extremely useful and would definitely bring to future projects that I work on.
- **Design:** For the design phase we researched different options and decided what architecture and technologies we were going to use. One of the big criteria was to give the application a "real-time" feel to it. So when you make a change to a data point in one place it automatically propagates those changes to derived data and other users. A bit like working on a collaborative Excel spreadsheet.
- **Build:** Then we started actually building the application. Once we has some user interfaces we started doing user testing. This allowed us to use an iterative design process and make sure we were building what our users actually wanted.
- **Parallel testing:** After the main features were built we entered a period of parallel testing. During this time we were running the new system in parallel with the old one, validating our calculations. As this was a financial application that would eventually send data to clients a high degree of certainty in the data was required. We also started with full scale user testing bringing on feedback from more people on the team. Of course this led to many bug fixes and some minor changes.
- **Production:** Finally after we were satisfied with our system we turned off the old system and Atlas took over.

Next slide

Atlas - user feedback

As you can imagine we received a lot of feedback from users during this project and this has greatly influenced how I think about designing software. For the most part the feedback we received on features was relatively obvious and easily resolved. There were two things however that have really stuck with me from this time, mostly because of how much controversy and discourse they stirred up.

Show next step. Firstly, pie charts. One senior member of the team was very much pro using pie charts extensively while some other people were vehemently against them.

This led to a whole discussion on the pros and cons of pie charts and when to use them. In the end we settled on using them sparingly throughout the application in places where they really made sense. Fortunately this discussion came up during the specification phase rather than at build so I didn't have go around re-writing any code. This was a big bonus of having visual references in the specification.

Show next step, the second one is how to align numbers in a table. The application made extensive use of tabulated data and one time, in a user review, I was asked "why are all the numbers right aligned, they should be centred".

Show next step, In other words why was I doing this, I should be doing this.

At the time I was shocked this was even a question but when I brought the query to a wider group of users I found people very strongly entrenched on both sides. In the end we made it a user setting so that everyone could set it up the way they wanted.

What I learnt from these two instances, and countless others, is that what makes a good user interface is extremely subjective. You should make configurable UIs so that users can set them up the way they want them. Especially in cases where your users are going to be using the application every day. Though I still think it is important to have sensible defaults so that new users can get up and running quickly.

There are really no right or wrong answers when it comes to UI design. So you should get your designs in front of users as early as possible to get their feedback.

Everyone will give you feedback on a design, even if they don't actually use the application (this often includes the developers of the application). So you should prioritize actual users over others where appropriate.

Next slide

Atlas - my contributions

The main things I contributed to the Atlas project are:

- I setup the continuous integration/continuous deployment pipeline. This allowed us to quickly and easily integrate changes and release them to users.
- I setup the C# and React projects and integrated them so they would work together both in development and production. Making the developer experience of the application seamless.
- I built the event system that automatically propagated data changes. This was one the most difficult technical challenges of the project as it involved building a custom queuing and event handling system.
- I built the real-time system that kept the frontend up to date using web sockets (SignalR)
- I built reusable components for the frontend. Several screens looked very similar so these components were used across all of them to keep the user experience consistent.

Next slide

Atlas - results

The Atlas project was a great success.

- We made a 20x performance improvement to the core calculation engine, going from over 20mins in the old system to just under 1min.
- We had a completely re-designed modern user interface that was much easier to work with.
- We now had real-time updates that synchronised across all users. Previously they would have to manually trigger the calculations and then wait 20 mins and then re-fresh their screens to get the changes.
- Most importantly we saved our users time and made their jobs easier.

Next slide

Atlas - why is this relevant?

So why is this relevant to what I will be doing at the Allen Institute?

This project gave me invaluable experience in delivering a technical software project. I feel like this broad experience will allow me to contribute to a project at any stage of it's development.

For this project I was working closely with subject matter experts to deliver them a product they will use everyday. In this case I was working with accountants and financial markets experts, but I'm sure this will translate very well to working with neuroscientists.

I was working closely with the users of an application. I learnt a lot about how people interact with software and how you can't assume anything is obvious in UI design.

Any questions before I move on?

Next slide

Rust ray tracer

Now we get to talk about a more interesting, albeit less useful project that I built. This was a passion project to build a ray tracing engine from scratch.

I've always been interested 3D graphics, mostly thanks to video games but I became interested in ray tracing in particular when I took a module all about the math behind it in college. To me it was a fun blend of math, physics, and computers, all of which are I am very interested in.

Later when I was learning Rust (a new programming language) I was looking for a good compute intensive project to test my skills and ray tracing seemed like a good candidate.

It's based on the 'Ray tracing in one weekend' book series, which takes you through how to build a simple ray tracing engine from scratch in C++, though my project was all written in Rust. It runs on the CPU so it is not very fast, but given enough time it can produce some cool images.

Next slide

What is ray tracing?

Something I should explain in case you are not familiar is; what is ray tracing?

Ray tracing is a method for generating images of digital 3D scenes. It works by sending out rays from a camera for each pixel on the screen. These rays then bounce off objects, sometimes multiple times, before eventually ending at a light source (or not).

Describe diagram. (As we can see in the diagram a ray coming from one of these two pixels will scatter off the ball and hit the light. Other rays coming from this pixel will scatter off the ground and be blocked from reaching the light source by the ball creating a shadow in the image.)

Typically multiple rays are sent per pixel all scattering in different directions. You can imagine that of the rays being scattered off this ball only a small number are going to end up hitting the light. We average all of these rays to determine the colour of each pixel.

In more complex scenes rays might interact with multiple objects before hitting a light source creating complex interactions in the image.

Ray tracing is very computationally intensive compared to other methods of rendering an image like rasterization. It prioritizes image quality over speed. Ray tracing is able to produce photorealistic images. Thus it is commonly used in CGI and visual effects, and sometimes even in video games, though it is often mixed with other techniques for performance reasons in that case.

Next slide

Ray tracer image

So this is an image produced by my ray tracer. The dragon you see in the middle is the 'Stanford dragon' which is a common 3D model that is used to test 3D software. It has about 900,000 triangles.

Here we have a glass ball. Note how it inverts the image that we see through it and how it focuses the light on the ground. This is what I mean when I say ray tracing is able to produce photorealistic effects. These kinds of details are very hard to achieve with other rendering means.

A few other things we see are a metal ball here which reflects other objects around it.

We also have a moving ball over here which exhibits motion blur.

Next slide

Move to WebGL tab

WebGL project

The next project I want to talk about is somewhat related. When I found out about this job I wanted to familiarize myself with WebGL so I decided to recreate the same scene from my ray tracing engine in the browser using WebGL (three.js). Which we can see here. This one uses the GPU and classic rasterization for rendering so we can interact with it!

You'll notice that some objects here are still interacting with other objects in the scene, like this metal ball. However this is not because we are bouncing rays around off of multiple objects.

In this case I have setup something called an 'environment map' for the ball. To do this we essentially take a snapshot of the scene from the perspective of the ball and then attach it to the ball as a texture. This is very similar to how we are wrapping the image of the earth on this other ball to make a globe.

This technique has it's limitations however. Notice the glass ball no longer looks very realistic, this is because we are no longer actually simulating refraction of light just making the sphere transparent with some distortion.

Also notice the green reflections on the roof of the dragon's mouth, these should be blocked by the lower jaw but this is not taken into account in this simple environment map.

This project came together really quickly, I was able to complete it in just a weekend. I attribute this to the skills that I learnt while building the ray tracer and my experience in web development.

Next slide

Why is this relevant?

So why is all of this relevant?

I think that building a ray tracing engine from scratch has given me a great foundational knowledge when it comes to 3D graphics that will translate to working anywhere in this space. This was demonstrated by my ability to quickly get up and running with WebGL.

Combining this knowledge with my previous experience of building user interfaces and web development I think will really enable me to excel in this position at the Allen Institute.

Any questions before I move on?

Next slide

Open source work

Something else I want to talk about briefly is some of my open source work. I am passionate about open source development, I think it is an opportunity to give back to the communities that I am a part of as well as a great way to learn new skills. The open science approach of the Allen Institute is one of the things that has really drawn me to the institute.

Next slide

Move to Dry Rock tab

Dry Rock

Dry Rock is an open source project I started about 5 years ago. It is a weather website for rock climbers. The main problem it is trying to solve is "where should I go climbing this weekend"?

With rock climbing, as with any outdoor activity, the weather plays a crucial roll. In particular with rock climbing, rain can really put a damper on your day out. This often led me to checking the forecast for multiple different places, trying to compare them to each other and figure out where was going to have the best weather. Thus Dry Rock was born!

It is designed for high information density so that you can quickly figure out where the best weather to go climbing is with the least amount of effort. It is also specifically designed to work well on mobile devices because that is often how I end up using it.

Demo Dry Rock

Although I originally just created Dry Rock for myself several of my friends and others in the climbing community are using it too. It's open source so people can provide feedback and raise issues on GitHub. So far the main feedback I've received about it is simply to add more places. When people want more of something I generally take that to mean it's doing a good job.

Something I have added to this recently, which I think demonstrates some of the things I have learnt since I first started this project, is having multiple ways to do the same thing. One such example of this is when I open this detailed view there are in fact three different ways of closing it.

- I can click this button
- I can click anywhere on the screen that is not part of the modal
- I can also hit the back button

Now no matter what an individual user thinks is the most intuitive it should work for them. This is an example of building flexible user interfaces that I referred to earlier.

Next slide

NWAC - Avy

I have also been volunteering for the Northwest Avalanche Center since January, helping them maintain their Avy app. The app provides avalanche forecasts and weather data from various avalanche centers in the US. It is used by thousands of people for backcountry skiing and other winter activities.

As a backcountry skier myself, this app is relevant to my life. I find that helping out with it is a great way to give back to the community and an organisation that is trying to keep us all safe in the mountains.

So far I have mostly been fixing bugs, especially on the Android app, as the rest of the team mostly uses iPhones. This is my first experience working on a mobile app, but it's been great getting to dabble in this area. As the app is mostly built using TypeScript I have been able to pick it up quickly.

Any questions before I move on?

Next slide

Why am I interested in the Allen Institute?

Hopefully I've done a good job of explaining how my experience will come in useful at the Allen Institute. However I want to talk a little bit about why I am interested in working at the Allen and why this job in particular.

As I mentioned before I am a big fan of open source software and I think this really aligns with the Allen's open science approach. Being a part of a team that creates software not just for people here but scientists all over the world is very appealing to me.

Additionally, as is probably clear from my ray tracing project, I am very interested in 3D graphics and the opportunity to work in that space professionally is not one that I could turn down.

A little story to finish off my talk. A few years ago when I was applying for my first job as a software engineer, my then girlfriend, now wife, was doing her master's degree in neuroscience and showed me the Allen Brain Atlas (which she was using in her project). I remember playing with the 3D visualisations of the atlas and thinking how cool it would be to create software like that, something that was totally unthinkable to me at that time. Looking back on this now I have come a long way and hope to make this a reality.

Next slide

Questions?

Thank you for your time, any more questions for me before we finish up here.