- Welcome to CSC301!
- Who am i?
    - Jason Wang
    - Lifelong engineer, started when i was 12 years old
    - Graduated from the university of waterloo ~10 years ago
    - In startups all my life, and now I am an engineering manager at Uber
- What can you expect?
    - My first time teaching the course - but I was a student once too
        - Very open to feedback on what you want to learn and hear about
        - No term tests, because I believe this course is fairly subjective and the main goal I have is to teach you about tradeoffs. There is rarely a one “right” answer, there are just degrees of pros and cons. Tests presume there is one right answer to everything.
        - Want to leave plenty of time for question and answers every lecture
    - Learning the practice of engineering software in real life, goal is to expose you to some of the messiness of building software and will touch on as many topics as I can
    - Be the singular most useful course you take in your university career.
    - I don’t want to speak for the entire 2 hours that we have together. Therefore I will break up the lectures into 1.5 hours in the beginning, and 30 mins at the end of open Q&A. You are free to come and go at any time, if you are interested in asking questions or listening to answers by me or your peers, feel free to stay for the entire 2 hour session.
- Project
    - This course has a huge project focus, personally the part that I enjoy the most is building something from nothing and building something not just for the sake of building it, but have it solve a real problem that someone has
    - The project will be done in groups, and it will simulate what a real project in a company might operate: you’ll be using real-world agile practices and proper git flow
    - One **Note**: projects in this course will have a strong utility requirement. The ideal end result will be viable as a starting point of a startup company. So no projects strictly for teaching purposes like a pet sitter project, calendar app or something generic pulled from another website
    - Project will be 60% of the course mark.
    - Components:
        - Prior to you starting the project, it’s required to write a one page proposal that (ideally) includes the following information:
            - A detailed description of what you are proposing to do
            - Some competitor analysis, with at least 5 competing services and go into some details about why what you are building is needed in the marketplace
            - Market sizing analysis, how big is the addressable market and who are you targeting in various stages of release. This can and

should go above and beyond what you are building during this course
        - Once development start, we will provide you with a detailed marking rubric
        - Sprints are one every 2 weeks, 4 sprints in total, there will be a few things you are required to do each and every sprint in addition to making progress on your development, such as sprint retroactives and keeping your Trello board updated
        - You'll spend the first week finding a team, and then you'll submit your sprint 0 deliverables (including the project proposal) at the end of week 3. Sprint 1 will start on week 4 and end at the end of week 5, etc
- What makes a good team
    - Two ways of finding fellow team members
        - People who compliment your skillset + with whom you work well together
        - People who want to work on the same idea
    - I want to encourage you to branch out and work with people who you don't normally work with
    - **Note**, just because you are on the same team together, it **DOES NOT** mean you will get the same grades
    - We will take a holistic approach to grading projects, and take into account peer evaluations (which are not public, so anonymous to your group members, but not to me and TAs), Github activities etc. As a result, individuals may get higher than group grade and vice versa.
    - However, I don't want this to be zero sum either. If everyone works hard, they'll equally share in the rewards, and I want you to help each other to improve in this course as well.
- Assignments
    - Assignments will be in Java and it will expose you to aspects/technologies that you may not otherwise be working with
    - I won't be the best person to ask questions about the assignments, as these assignments are inherited from previous iterations of the course. Feel free to ask any of the TAs about the assignments though during tutorial or on piazza (you *can* ask on discord, but try to keep assignment-specific questions on piazza). Some of the TAs will also be hosting assignment office hours throughout the term.
    - Assignments will be 15% and 20% of the marks respectively, due Feb 18th and April 05.
- Tutorials
    - There will be a total of six in-person tutorials which will be graded for participation. In this course, as long as you show up for the tutorial (and aren't doing some other work unrelated to the tutorial), you will get full marks. We will be only counting your "best" 5 out of the 6 tutorials. This means that you can miss one of the tutorials and still get the whole 5%. More details will be discussed during your first tutorial.

    - The first tutorial will take place this week on friday january 13th
    - Tutorials will be primarily focused on introducing and teaching new technologies to you such as Neo4j, docker, mongo, react, etc. You will need to use much of the content covered in tutorial within your assignments, and you may also would like to use some of it in developing your group project.
- How to ask questions
    - It's never acceptable to ask "hey, ____ doesn't work, help!", if there is no evidence you have attempted to solve your own problems, your questions will likely be ignored.
    - A better template to use is
        - Hi everyone, I am working on ____, and I am stuck on ____. I tried ____, it didn't work because _____. I also tried _____, and it didn't work because ____. I am assuming _____ and ____. Does anyone have any hints as to what might be wrong?
    - Unless it's a persona/private question that involves your personal information, please post any and all questions into Piazza so that everyone may benefit from any answers that come from it.
    - My office hour will be 1 hour immediately following the lecture. If you want to be guaranteed a slot, please make an appointment either by email or before the lecture. Otherwise I will try my best to meet with everyone who is interested in talking to me.
    - When sending messages on Discord, never just say "hey" or "good morning" and wait for the other person to respond. Always include your question in your first message, so that when the other person sees the message, they can immediately compose an answer to your question.
- Difference between real world and academia:
    - 70% of the stuff you learn at university you will not use 90% of the time in real life.
    - For example, you will not be designing new and novel algorithms, the data structures you will be using consists of array/list/dictionary (basically everything that comes with JSON). You will not be thinking about the network stack at all, or details about operating systems etc.
    - Why are you learning it?
        - Because of leaky abstraction: https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/
    - 80-90% of the time, you are operating on a abstraction level so far above anything difficult or academic that what you are learning feels like useless and a waste of time
    - However, the 10-20% of the time when shit hits the fan is when your 4 years of education pays off compared to someone who came out of a bootcamp for 4-6 months.
    - That depth of thinking and understanding is why you are all sitting here listening to me speak, the days and nights struggling with assignments and studying for tests. So that one day, when you are oncall at a company and something goes

wrong, you are/feel equipped to tackle the problem regardless of where things go wrong
    - You at least understand the underpinning of what makes computers tick all the way from processor design/1s and 0s all the way up to whatever framework dejur that you are using
- Software development models
    - Software is interesting, traditional engineering (for bridges and planes etc) requires a great deal of planning and requirement gathering ahead of time. The amount of prep you spend is directly correlated with how expensive it is to fix mistakes when in “production”.
    - Imagine an extreme example, NASA is launching a 10 B rocket to explore the far fringes of the galaxy. There is pretty much zero opportunity to fix issues once the payload is launched, it needs to work the first time or you just wasted 10B. Therefore, if you can write 10 lines of code per day that go into the final product, you should consider yourself lucky. Every line that gets written, every component that gets built, is carefully considered and scrutinized.
    - Imagine another extreme example, you are building a quick script to solve a pressing problem of your own. As long as you control all the systems, it doesn’t really matter if something breaks. You are the author, you can fix whatever that comes up. As long as it works for the case you are considering, it’s good enough to launch.
    - You would never apply the software development model used for building a rocket to your personal project, and vice versa. **Professional development there is never a right answer, just degrees of tradeoffs that you as a professional are making.** My job here is to start you down the path of critical thinking and as a result, tests really have no place in this course.
- Two primary software development models are **Waterfall** and **Agile**.
    - Traditional view are that Waterfall is an old and antiquated way of doing software development. Typically derived from traditional engineering disciplines such as civil and mechanical. Where we go through distinct phases of software development from requirements gathering, design, implementation, and verification.
    - Agile is the new and shiny way (and typically what you would find in most software development organizations). The core of it basically states the traditional waterfall ways doesn’t yield useful results, it’s not responsive to changing customer and business requirements. Agile is much more focused on incremental development, short cycles and the goal of always having working software as much and as often as possible.
        - Backbone process: Sprint, typically 1-2 week. Any longer than that opens up the sprint to be much less flexible and less responsive to changing needs
        - Each sprint typically bookended on both sides by a planning session and a retrospective session. Where planning the team talks about what will be

done the next sprint, and retrospective typically talks about what worked and what didn't work

- It also typically includes daily standups, where the main purpose of the standup is for everyone to know what everyone else is working on, plus for people on the team to voice any blockers that they are encountering and ask for help from fellow members to help unblock.