Slide 1:

I am Paras and I will be presenting Ashoka cabs. This app is being developed in collaboration with Ashoka IT department. I will start with briefly describing the app and then we will the live application deployed on AWS server.

Slide 2:

As we know, especially when semester starts or ends, we have so many vehicles from nearby places coming to and from campus. Many students start sending mails to have a companion while traveling to common places like airport. Many people are using private vehicles while others are using Ola or Uber. Our value proposition is to use these vehicles for carpool, which might help people connect over the trip to campus and save money, and environment.

We try to achieve this by connecting those who have vehicles to those who are planning to commute, with an easy to use web interface. Hence introducing Ashoka Cabs, a listing portal which helps you see and book seats in vehicles, specifically to Ashoka.

Slide 3:

Following our value proposition last semester, we started designing a web application for carpool, following Software Development life cycle. After analysing market need, the cycle starts with gathering specifications for the app. After requirement analysis, we moved to design the app, which included creating wire framing with getting user feedback on design. Following which we moved to implement the app using the available resources, taking decisions on which language to use, which database fits our requirements more appropriately.

Last semester basic implementation was done, hence this semester we started with testing the system, involving both white and black box testing. In case of white box testing, the tester knows the internal implementation of the app, like the language and database specs we are using, which was done with the help of Ashoka IT team. For Black box testing, the system is tested in the condition it is available to the users, which was done with friends and other Ashoka IT members.

This semester the major part was evolution and testing, with the help of Ashoka's IT team and friends' testing and goal was to make making the app more usable.

Slide 4:

Starting the development life cycle, we started with gathering requirements. So we had two main stakeholders for our app. Getting all the requirements from students would have resulted in one thousand four hundred features, assuming each one contributing one feature. So for the first prototype we decided to gather requirements from admin and then validating those from students.

Requirement specifications for the first version were curated by Ashish sir who was in charge of shuttle operations, Anu ma'am and Turam from IT department, in guidance of Professor Hangal.

We got 242 responses from students and 48 from faculty for validating the requirenments for the first version of our app. Then to finally start with the first version, we started building wireframes as we can see in upcoming slides.

Slide 5:

These are the screenshots of the first wireframes that I developed for the app.

For developing wireframes I used wireframe.cc , free and simple online tool.

Slide 6:

Initial design of the add vehicles and my vehicles page. We started with this but have come a long way.

Slide 7:

These wireframes were revised several time with feedback from admin and Professor, along with ASP student Rehan, Rhea, Sankalp, and myself.

After this we started implementing the system.

Slide 8:

We wanted to experiment with new technologies, so we decided to use Javascript on backend, using Node.js as JavaScript runtime on server. So we have JavaScript on front-end, and on backend. We used ejs for templating, mongodb for database along with Passport.js for gmail authentication.

We have kept the code base open sources.

In case you want to contribute to the code base, you can visit the link above and fork the repository.

Slide 9:

In the first version, for the first development lifecycle, we implemented very basic endpoints to make it functional. The endpoints are

/user is for getting user details belonging to email, above apis are not open, can only accessed within the application

/user/vehicles is for listing all the user vehicles

/user/vehicles/add is when user adds a vehicle for car pool

/user/bookings is for checking all the bookings user has made in other vehicles

/user/bookings/add is for adding a seat

/user/bookings/cancel is for cancelling the seat

/vehicles for listing all the vehicles going from source to destination at a particular point in time

Slide 10:

Last semester we were not able to complete because testing was remaining. So this time we started with testing.

First step was Interface and Usability testing which was achieved with the help of friends. Few of us sat down together and was asked to add a vehicle and make a booking without any guidance. There experience gave us inputs for improvements.

Since we needed to test the interface multiple time with multiple people with all the endpoints mentioned in previous slide. We created a testing document which enlisted all the apis and steps to test them along with expected behaviour. This sounds easy but actually was very helpful as testers also found written instructions easy to comprehend.

For Security testing, we did penetration testing, and ended up sanitizing inputs. Thankfully unlike JSP, EJS provide methods to sanitize untrusted strings without any other dependency. In java we have to use StringEscapeUtils, but ejs it can be done without any dependency.

Compatibility testing includes testing it on various devices. We optimized mobile interface for 5.1 inch android screen after some users tried our interface on phone.

Although it also works well on other screen sizes, but best on 5.1

For Load Testing, we developed a program that spins multiple threads. Each thread then books and cancels seats in a test vehicle.

Slide 11:

After many meeting with IT department and changes suggested by users who have tested the app, we added the following end points:

Edit api came around after following a bug. So we had conditional check to see if the user has filled the profile, if yes then the profile page re-directs user to make bookings page, but then we realized, that the conditional check was preventing the user to change the profile credentials, every time the user visits the profile page, re-direction was happening. To solve this problem, we have to create this new endpoint on same page, but with get request this time. So next time you fill the form and do a post, you will be re-directed, but if you click the button to change profile, you will hit this edit end point.

Cancel endpoint allows user to cancel a seat, which was a feature in demand

Remove endpoint is to remove vehicle which has zero ocupancy

Slide 12:

-Apart from navigation and message enhancements like dialogue for ‘Profile being saved successfully’, The main enhancements we came up with are as follows, we added waitlist option on every vehicle, with counters in negative to indicate the number of passengers in waitlist.

-Removing price fields was a decision made to help us differentiate from other cab services.

-Autofill part through javascript although was a mojor step, but was really helpful as the small form is pre-filled with current date and time range which might be the common usage case for most users, which was the case during testing and was highly recommended by IT and our test users.

- No point in adding wailist if we don’t have waitlist promotion, in case some one with confirmed booking cancels the seat, the first person in wailist will be promoted to confirmed status
- Email notification using Gmail API, now user gets email everytime the booking is confirmed, even when the confirmation is through waitlist promotion
- Although adding vehicle on a past date doesn’t makes sense, but still some people did that, so we have to block it programmatically.
- -In first iteration we thought only a confirmation email would be sufficient as the booking can be identified through email ID, but we realized booking ID is useful in case of multiple bookings by the same user, which we may be used to allocate seat numbers in future. Also, booking ID will help in case some one forges email from App’s email account.
- These enhancements has been added as a result of multiple meetings throughout the semester with IT department, Some of them with Ashish sir who left around mid sem. With each round of enhancement, we had to repeat the testing process. Although it sounds

redundant to test after each improvement because many features are mutually exclusive, but it is really helpful as I will share some experiences.

Slide 13:

So the first time we received a bug, it was about not receiving the mail after booking. This was because the Gmail API tokens expired. We literally had no way to figure that out. So we set up nohup(node) logger which logs every error. Unlike Tomcat, this was not a default feature of node server. After setting up logger, it was relatively easy to figure out the bug in code

Testing after each step made sure that we are not compromising the usability. Also, once, we updated the views on server but forgot to update the js files related to mail API, hence email again stopped working. If we hadn't tested the entire app again, we wouldn't have been able to find the mistake on the email part as it was untouched in the new feature.

App is being developed on a window machine where IDE mostly takes care of things like error logging, managing downloaded dependencies but the deployment environment is in linux machine with no IDE. It happened multiple times that linux machine didn't downloaded dependencies because while downloading dependencies on windows machine, dependency name was skipped being added to the package.json, which is a configuration file necessary to download right dependency on machine without any IDE.

So being close to production environment during development is much recommended, at least from my little experience of building this app.

Client side bug sometimes include disabled javascript in the browser. Which are very difficult to detect because that cannot be logged under any exception handler on server logs. In this case, disabling javascript blocks the form to autofill, thankfully this happened while testing so we now the cause. Always assuming that the code is not working because the logic is mainly on server side can be wrong.

Slide 14:

You can see our first prototype at the above address. We have uploaded the shuttle schedule for next 5 months in our database as we are trying to use the app for managing shuttle as well.

Challenges include staying close to wireframes, as you start writing code, you discover various different features that can be added, specially in UI. Node is open source and has huge dependency database where people contribute. It is very easy in this world to get the wrong dependency and break everything, even the security. For example, the famous one is crossenv with hypen and without hypen. If you ignorantly downloaded the malicious one, it will work as per the feature specifications of the non-malicious one but at the same time, makes your code vulnerable to hackers. So people are trying to destroy the trust in open-source market.

Each time we use the app, we can see things that can be improved or added that we could not have seen earlier because the feature doesn't existed or we get new ideas.

Incorporating every single feature in code is an implementation challenge.

Thanks.